laSalle ENG
Ramon Llull University

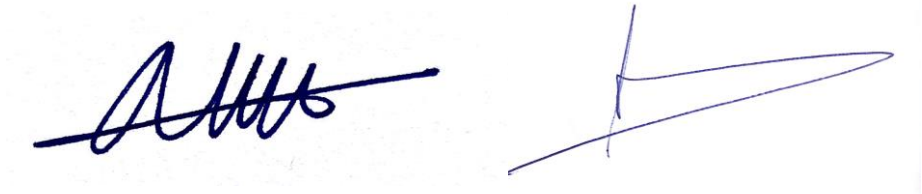
Escola Tècnica Superior d'Enginyeria Electrònica i Informàtica La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Telecomunicació

Desarrollo y análisis de resultados de herramienta para el estudio de la privacidad y monetización de la web.

Alumne	Professors Ponents	
Albert Terradas Moreno	Miguel Ramírez Martín	Rosa Maria Pagès



ACTA DE L'EXAMEN
DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D.

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Tabla de contenido

Tabla de Figuras

Tablas de Datos

1. Abstract

From the initial static websites, until the current web applications, browsers have evolved to the point of implementing functions that were previously characteristic of the operating system, that make the browsers more powerful but at the same time can imply undesired situations for the user.

The malicious use of these components allows to identify the devices where they are used, with the objective to trace user behavior, and leads to what is known as the techniques of **fingerprinting** still a quite unknown practice.

The birth of the **cryptocurrencies** and the possibility to mine them in a distributed way, has led to risks to the user as their computational power can be used by third-parties with the objective of generating income.

Therefore, it is intended to **identify the use of these techniques** in an environment intentionally obfuscated by their creators.

The main aim of this project is the **analysis of the trends** in the use of these techniques and their effect on the privacy and security of the users, through the monthly examination of hundreds of thousands of websites.

The second objective is the **diffusion of the results** obtained on open source through the creation of: a database with the information and one API to consult it, a website for the average user with information on fingerprinting and the sites which are using it, lists that allow to block the scripts that use them (useful for third-party applications) and an extension for Google Chrome that warns about the use of these techniques when visiting a website.

This study has been realized at Telefónica I+D, through the development of a scholarship granted by the company to the University of Princeton (New Jersey, USA) and in collaboration with the University of Lehigh (Pensilvania, USA).

Resum

Des de les primeres pàgines web estàtiques fins a les actuals aplicacions web, els navegadors han evolucionat fins a implementar funcions que prèviament eren pròpies del sistema operatiu i que fan molt més potents els navegadors però que poden dur a terme a situacions no desitjades comprometent a l'usuari.

L'ús malintencionat d'aquests components permet identificar amb molta precissió els equips en què s'utilitzen i donen lloc a allò que es coneix com a tècniques de **fingerprinting**, una pràctica de rastreig de usuaris encara molt desconeguda però cada cop més utilitzada.

El naixement de **les criptomonedes** i la possibilitat de minar-les de manera distribuïda ha comportat també riscos de cara a l'usuari on, a més, la seva potència computacional es utilitzada per tercers amb la finalitat de generar ingressos.

Per tant, es pretén **identificar l'ús d'aquestes tècniques** en un entorn intencionadament ofuscat i poc transparent per la pròpia naturalesa del objectiu que pretenen els seus creadors.

L'objectiu principal d'aquest treball és **l'anàlisi de les tendències** en l'ús d'aquestes tècniques i el seu efecte en la privacitat i seguretat dels usuaris, a través de l'examen mensual de centenars de milers de llocs web.

El segon objectiu és la **difusió dels resultats** obtinguts de manera oberta realitzant: una base de dades amb l'informació i una API per consultar-la, un lloc web per a l'usuari mitjà amb informació sobre fingerprinting i els llocs on el realitzen, llistes que permetin bloquejar els scripts que les utilitzen (útils per a aplicacions de tercers) i una extensió per a Google Chrome que ens adverteixi de l'ús d'aquestes tècniques en visitar una pàgina web.

Aquest treball ha estat realitzat a Telefónica I+D, en col·laboració amb equips de treball de les Universitats de Princeton (New Jersey, EUA) i de la Universitat de Lehigh (Pensilvania, EUA).

2. Resumen

Desde las primeras páginas web estáticas hasta las actuales aplicaciones web, los navegadores han evolucionado hasta implementar funciones que previamente eran propias del sistema operativo y que hacen de los navegadores, herramientas mucho más potentes pero que al mismo tiempo pueden llevar a situaciones que comprometan al usuario.

El uso malintencionado de estos componentes permite identificar los diferentes dispositivos donde son utilizados, con el objetivo de rastrear el comportamiento de los usuarios, y dan lugar a lo que se conoce como técnicas de **fingerprinting**, una práctica de rastreo aún muy desconocida y cada vez más utilizada.

El nacimiento de **las criptomonedas** y la posibilidad de minarlas de manera distribuida, comporta riesgos para el usuario, y además, su potencia computacional es utilizada por terceros con el fin de generar ingresos.

Se ha pretendido identificar el uso de estas técnicas, en un entorno intencionadamente ofuscada y poco o nada transparente, por la propia naturaleza del objetivo perseguido por sus creadores.

El objetivo principal de este trabajo es **el análisis de las tendencias** en el uso de estas técnicas y su efecto en la privacidad y seguridad de los usuarios, a través del examen mensual de centenares de miles de sitios web.

El segundo objetivo es la **difusión de los resultados** obtenidos de manera abierta, realizando una base de datos con toda la información recogida y una API para consultarla. Un sitio web para el usuario medio con información sobre fingerprinting y los sitios web que lo utilizan, listas que permiten bloquear los scripts que utilizan (útiles para aplicaciones de terceros) y una extensión para Google Chrome que nos advierta del uso de estas técnicas al visitar una página web.

Este trabajo ha sido realizado en Telefónica I+D, integrado en un proyecto conjunto con equipos de la Universidad de Princeton (New Jersey, EUA) y de la Universidad de Lehigh (Pensilvania, EUA).

3. Introducción

Este proyecto ha sido desarrollado en Telefónica I+D tomando parte en el departamento del Data Transparency Lab (Barcelona). El propósito del DTL es crear una comunidad global de investigadores, políticos, reguladores y representantes de la industria que trabajen para avanzar en una temática en auge como es la transparencia de datos personales en la red mediante la investigación científica y el diseño que permita dar visibilidad a esos resultados.

Por lo tanto, este proyecto nació con la intención de analizar métodos poco transparentes de identificar al usuario y ha evolucionado hacia el análisis de nuevas tendencias cómo el minado de criptomonedas mediante scripts ejecutándose en el cliente dadas las posibilidades de la herramienta implementada.

3.1 Planteamiento del problema y actuación del DTL

Desde el inicio del www, se ha intentado obtener una rentabilidad comercial. En la fase inicial, utilizando la funcionalidad básica de comunicación y para compartir información. Esto rápidamente evolucionó hacia un escaparate gigantesco.

Una vez teníamos el escaparate, apareció el negocio de dirigir a los consumidores hacia el mostrador de quien estuviese dispuesto a pagar por ese servicio. Este servicio "añadido" se convirtió rápidamente en el servicio "esencial" alrededor del que gira la mayor parte del negocio de las empresas denominadas "digitales". Las empresas que son capaces de captar a los usuarios, normalmente ofreciendo servicios colaterales de manera gratuita, conocerlos y dirigirlos a la vitrina del anunciante adecuado, están generando un negocio importantísimo y, hasta hoy, sin límite aparente.

Para "conocer" al usuario/consumidor, estás empresas utilizan multitud de estrategias basadas en la captura y análisis de sus datos y sus intereses.

Naturalmente, esta utilización de datos de las personas (en síntesis, información sensible), que ponen en cuestión su privacidad, preocupa de manera muy importante y creciente a la sociedad, y por extensión a los reguladores y a los gobernantes.

Es sabido que:

- Para solucionar un problema, el primer paso, es entenderlo (George Pólya, 1945).
- Todo lo que se puede medir, se puede mejorar.

Y este es el objetivo del DTL, desarrollar y ofrecer herramientas que proporcionen un conocimiento exhaustivo de las distintas técnicas que se utilizan para obtener datos personales, y su grado de implantación. Estas herramientas deben ser dinámicas y avanzar al ritmo de la industria.

Para ello, se estimula la cooperación internacional, financiando proyectos académicos

en las universidades más prestigiosas del mundo en el campo de la privacidad de datos, como son las universidades de Princeton, Northeastern, Carlos III, Stony Brook, etc.

con el objetivo de desarrollar proyectos que detecten y den a conocer todas las practicas, incluidas las más recientes, en el campo de la privacidad de datos en la red.

El DTL junto con Mozilla y los miembros cofundadores, MIT Connection Science, Max Planck Institute, ATT, Telefónica y otros miembros colaboradores como son Orange y Inria, es el encargado de difundir esos resultados y realizar las mejoras en los proyectos que se consideren necesarias para que se cumplan el objetivo de dar visibilidad a las técnicas utilizadas para la obtención y las formas de uso de los datos personales obtenidos, para así, en caso necesario, los reguladores, la industria y los usuarios puedan tomar las medidas que consideren apropiadas para proteger su privacidad.

3.2 Visión global y objetivos del proyecto

Este proyecto se concibió con el objetivo de complementar y añadir nuevas funcionalidades al proyecto OpenWPM, becado por telefónica I+D, y desarrollado en la Universidad de Princeton por el estudiante de doctorado Steven Englehardt, bajo la dirección del profesor Arvin Narayanan.

OpenWPM es un framework para realizar medidas de privacidad en la web. Permite recopilar datos de privacidad en miles o millones de sitios web. OpenWPM funciona sobre Firefox, y la automatización que ofrece Selenium.

El primer objetivo del proyecto es el de complementar OpenWPM para que los resultados obtenidos sean entendibles y utilizables no solo por los investigadores, sino también por la industria, los reguladores, los medios de comunicación y también los usuarios finales. El segundo objetivo es detectar y solucionar posibles “bugs” de un proyecto todavía en desarrollo, poner de manifiesto e implementar funcionalidades todavía no incluidas en OpenWPM. En este se han analizado técnicas, poco estudiadas y estandarizadas, usadas para el rastreo de los usuarios de páginas web.

El objeto incial del proyecto ha sido el análisis del conjunto de técnicas conocidas como Fingerprinting y se han puesto las bases para una posterior extensión al estudio de las técnicas de Cookie-Syncing.

Para ello, todo el framework se ha diseñado para ser una herramienta escalable, a la que se puedan añadir al proceso nuevas técnicas de rastreo o medidas que afecten a la seguridad, y sus resultados añadidos a una base de datos única.

Finalmente, cómo he comentado previamente, la escalabilidad y versatilidad del estudio que nos permite la herramienta realizada ha permitido el análisis de una problemática muy actual y como consecuencia con un valor añadido, como es el minado de criptomonedas y que a priori no estaba contemplada como tal en el inicio del proyecto. Este análisis a día de hoy en tiempo real únicamente está siendo investigado por Brave

Software Inc. Empresa desarrolladora del navegador centrado en la privacidad del usuario Brave pero su solución aún está en fase de investigación, según comentó Ben Livshits, Chief Scientist de Brave, en la conferencia DTL2017 celebrada el pasado diciembre.

Por lo que también se ha realizado un análisis del segmento de las criptomonedas y en particular de su proceso de minado para de tal modo ser capaces de idear un sistema de detección.

Es necesario recalcar que la detección tanto de fingerprinting cómo de minado, a diferencia de todos los sistemas actuales, que se basan en detección de urls de script previamente conocidas mediante extensiones o plugins en el navegador o analizando el uso de la CPU mediante antivirus en el caso de la detección de criptomonedas, este método se basa en la detección en tiempo real por lo que es mucho más preciso y efectivo, tanto desde el punto de vista de detectar realmente los scripts que se están ejecutando (ya que puede ser que únicamente se estén cargando) como de la detección de nuevas implementaciones desconocidas.

A modo de resumen, los objetivos del proyecto son:

- Mejora de la herramienta de Web Crawling, para añadir funcionalidades y corregir errores.

- Recoger información de páginas web y analizar el uso que cada web hace de las técnicas de fingerprinting y minado de criptomonedas.

- Almacenamiento de la información obtenida en una base de datos estructurada.

- Diseño e implementación de la herramienta de análisis de resultados.

- Presentar los resultados de forma entendible y atractiva para diferentes tipos de usuarios.

- Dar la posibilidad a terceros de incluir la herramienta en su navegador, aumentado de esta manera su difusión.

- Desarrollar el software de manera eficaz con una integración continuada y un control de versionado de cada una de las diferentes herramientas.

3.3 Requisitos del proyecto

Los requisitos del proyecto han ido mutando en función de las necesidades del departamento, las posibilidades y oportunidades que han ido surgiendo en el transcurso del mismo.

En este apartado se contemplan los requisitos finales únicamente y no los requisitos intermedios necesarios para llegar a esos fines.

Las funcionalidades en la herramienta de Crawling necesarios para llegar a tales objetivos se detallan en los siguientes puntos de la memoria del proyecto.

Los requisitos finales son:

- Estudiar e identificar las técnicas usadas para llevar a cabo técnicas de fingerprinting y minado de criptomonedas.

- Hallar una “firma” que permita identificar las técnicas de fingerprinting y minado de criptomonedas inequívocamente.

- Obtención de la información de los sitios necesaria para cumplimentar el análisis.

- Reducir el consumo de memoria frente a la utilidad inicial y conseguir el máximo rendimiento.

- Diseño e implementación de la herramienta de análisis.

- Realización del test del censo de manera mensual.

- Aumentar el censo a 400.000 sitios web.

- Definir la estructura de las bases de datos (local y remota).

- Implementación de una API para la búsqueda directa en nuestra base de datos.

- Realización del análisis de los resultados almacenados en la base de datos para sacar conclusiones, en función de diferentes parámetros.

- Implementación de un sitio web donde se muestren los resultados de manera visual e intuitiva.

- Implementar en el sitio web la posibilidad de ver la información para un sitio web en particular.

- Análisis de evolución y tendencias para comparación entre diferentes periodos.

- Análisis inverso, hallar los scripts que rastrean más a los usuarios.

- Hallar una “firma” para identificar el minado de criptomonedas.

- Dimensionar el proyecto para incorporar nuevas técnicas más allá del fingerprinting como Cookie-Syncing, o el minado de criptomonedas en caso de ser viable su implementación.

- Generar información tanto para proyectos internos de Telefónica, como externos.

- Estudio de la viablidad de una herramienta que realice las mismas tareas directamente en el navegador y su desarrollo.

- Generar un backend para los logs de la herramienta en navegador.

3.4 Modificaciones e incidencias

Durante el transcurso del proyecto se han producido contratiempos, pero todos, de un modo u otro, se han solventado de manera eficaz y ninguno ha sido insalvable o ha retrasado enormemente el proyecto. Se han producido modificaciones durante el transcurso del proyecto, para cumplir con periodos necesarios para el departamento, siendo necesaria una primera versión en noviembre y ser presentada en el evento anual del departamento.

Durante el evento según el feedback recibido por especialistas del sector se decide realizar una segunda iteración desde el diseño hasta las herramientas y formatos del resultado.

Las modificaciones producidas son de carácter ampliatorio, realizando el análisis sobre censos mayores o modificando el tipo de listas elegidas para formar el censo, también las herramientas que hacen uso de los resultados se han ido perfilando a medida que el proyecto evolucionaba.

4. Herramienta de análisis

4.1 Framework OpenWPM

Para realizar medidas de privacidad en la web, es necesario simular usuarios, recoger los resultados obtenidos y analizarlos.
OpenWPM es una herramienta de código abierto que realiza de modo eficiente las dos primeras partes del proceso.
A diferencia de otras herramientas que solo pueden realizar medidas "stateless" (sin cookies), OpenWPM puede realizar medidas "stateful" (con cookies).
Ello permite obtener mejores resultados al analizar el ecosistema de trackers incluyendo cookie syncing, e incluso puede servir para analizar cookie respawning en el futuro.
Cookie respawning es una técnica mediante la cual los trackers almacenan las cookies del usuario y cuando el mismo usuario los vuelve a visitar, pero ha eliminado las cookies, cookie respawning permite regenarlas utilizando la copia del tracker. También son conocidas como "cookies zombies" porque renacen de las cenizas.
Al ser código abierto, OpenWPM permite implementar modificaciones y adaptaciones necesarias para el proyecto.

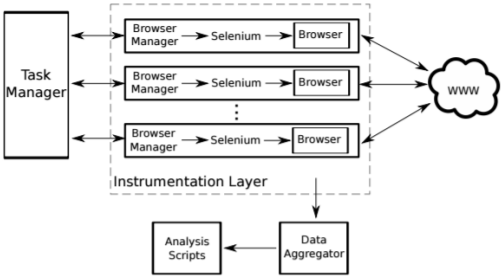


Figura 1 Esquema modular de OpenWPM

Esta herramienta hace uso de otras muchas entre las que destaca Selenium, para automatizar los navegadores como lo haría un usuario. Aunque Selenium soporta varios navegadores, se ha elegido Firefox para los test ejecutados durante este proyecto.

Además de Selenimum, para este proyecto se ha hecho un uso avanzado de plugins, y de Instrumentación Javascript, que se interpone y captura cualquier llamada a una API Javascript.

Se ha hecho un análisis de comportamiento de las "tracking cookies", los parámetros a analizar y adecuación de la base de datos para almacenar estos resultados en el futuro.

OpenWPM recopila todo el contenido durante una ejecución y lo almacena en una base de datos, en este caso SQL. La información obtenida, se analiza posteriomente mediante nuestros scripts de análisis, que acceden a la vista de la base de datos.

En el curso del proyecto, se ha creado un procedimiento para realizar medidas comparativas cuando se habilitan o deshabilitan plugins de protección contra el rastreo, tales como Adblock, Ghostery o Do Not Track. Esta funcionalidad, que se ha

desarrollado dentro de este proyecto, permite medir la eficacia de cada uno de esos plugins, y para cada sitio web o tracker.

En la tabla siguiente, se muestran varios estudios previos al realizado en este proyecto, que han realizado diferentes universidades usando varias de las posibilidades de OpenWPM.

Study	Type	Browser automation	Stateful crawls	Persistent profiles	Fine-grained profiles	Advanced plugin support	Automated login	Detect tracking cookies	Monitor state changes	Content extraction
HSTS and key pinning misconfigurations [28]	Security	●	●			●	○			●
FB Connect login permissions [51]	Privacy	●					●			○
Persistent tracking mechanisms [6]	Privacy	●	●	●	●			●	●	
Surveillance implications of web tracking [50]	Privacy	●	●			●		●		
Secondary authentication mechanisms	Security	●	●	●	●	●	●			○
News personalization ("filter bubble")	Personalization	●	●	●						○

Figura 2 Estudios realizados con OpenWPM previamente

Por lo que podemos concluir que es una herramienta estable y fiable para utilizar como base de nuestra herramienta de Crawling.

4.2 Modificaciones y mejoras

Durante el transcurso de este proyecto se ha ido iterando sobre esta herramienta para adaptar, mejorar y resolver bugs que se han ido encontrado, al ser una herramienta de experimentación hay detalles que hay que acabar de pulir, el servicio Github nos ofrece unas posibilidades muy interesantes ya que nos mantiene en contacto con la comunidad desarrolladora y podemos compartir las mejoras desarrolladas.

Los objetivos principales de la manipulación del framework inicial, son conseguir la estabilidad necesaria que nos permita la automatización total del proceso sin fallas que ralenticen y hagan necesaria la intervención humana con la consiguiente pérdida de recursos tanto temporales como de datos que quedan corruptos en determinadas situaciones.

Hacer mención que el objetivo nunca ha sido modificar este framework sino crear todo el entorno sobre él. Aun así, se han añadido mejoras que eran necesarias para realizar el proyecto de una forma adecuada.

Uno de los primeros retos era que el total de la información fuera procesada y analizando les resultados obtenidos de lso primeros tests vimos que las transmisiones comprimidas en formato brotli el crawler no era capaz de descomprimirlas. Esta compresión introducida por Google, está cada vez siendo más usada debido a que se adapta más a las necesidades actuales que el método más usado hasta la fecha, gzip. En las siguientes dos tablas se muestra una comparativa de ambos.

Archivo	*Tamaño(gzip)*	*Tamaño (brotli)*	*Reducción*
scott-test.html	0.763Kb	0.51Kb	33.15%
bootstrap.css	21Kb	19.07Kb	9.19%
image.jpg	203Kb	202.83Kb	0.08%
jquery.js	75.4Kb	70.67Kb	6.27%

Tabla 1 Compresión Gzip vs Brotli

Como vemos es un algoritmo que comprime más, por lo que se reduce el ancho de banda usado se cual sea el tipo de archivo a comprimir, es más exigente en origen ya que requiere más memoria, pero la compresión se realiza más rápidamente. Hoy en día donde el recurso más limitado es el ancho de banda, es un algoritmo que está ganando popularidad.
Investigando sobre porque OpenWPM no descomprimía brotli, cómo indica actualmente su documentación:

- NOTE: In addition to the other drawbacks of proxy-based measurements, content must be decoded before saving and not all current encodings are supported. In particular, brotli (br) is not supported.

Analizando el código vemos que la problemática se encuentra en la versión de Firefox usada por openwpm ya que la versión inicial de Firefox 42 no soportaba completamente esta funcionalidad. Viendo que la versión 45 sí que acepta esta codificación, se cambian los binarios. Una vez nuestro navegador acepta contenido brotli pasamos a la modificación del código de OpenWPM para aceptar tal funcionalidad, cómo hemos comentado previamente el framework está realizado en Python por lo que, al ser un lenguaje tan extendido, vemos que existe el package brotli con métodos de compresión y descompresión que nos permite de manera sencilla instalar en nuestro servidor la codificación brotli mediante el instalador de paquetes Python “pip”.

Usando el paquete podemos identificar y descomprimir los archivos brotli por lo que hará nuestro test muchos más fiable al analizar el total de la información y no perdiendo posible código usado para rastrear al usuario. En los test realizados vemos que brotli se decodifica perfectamente gracias a esta implementación. Esta mejora no se añadió en su momento a la herramienta original por la modificación de la versión de Firefox usada, actualmente la herramienta usa esta última versión de Firefox

Otro extra que se ha añadido al framework es la instrumentación para la detección de las llamadas a los métodos usados para realizar audio fingerprinting de la API audioContext. Esta instrumentación al ser parte de una técnica novedosa no se encontraba en la base de en la herramienta, al ser de interés general se hizo un pull request al repositorio de la herramienta original desde nuestro fork y han sido añadidas por el responsable del repositorio de la universidad de Princeton, Steven Englehardt, a la herramienta original. Los métodos que analizaremos más en detalle en el apartado de scripts de análisis, deben ser añadidos a la instrumentación ya que de otro modo no serán almacenados en nuestra base de datos para un posterior análisis. En concreto las llamadas que se han añadido para tal detección son las siguientes:

`AudioContext, OfflineAudioContext, OscillatorNode, AnalyserNode, GainNode, ScriptProcessorNode.`

En el siguiente apartado, de análisis se detalla el uso de cada método de la API para generar el fingerprint.

Por último, otra mejora que pasó desapercibida en los primeros test, han sido errores de formato en los datos obtenidos de sitios web. En concreto, la herramienta no está preparada por defecto para recibir valores erróneos de fechas de expiración en las cookies ya que la base de datos utilizada por la herramienta es SQL y, al crear la tabla definimos los campos de fechas como DATETIME de tal modo que espera el resultado exhaustivamente en ese formato (PostgreSQL, 2016) a diferencia de las bases de datos NoSQL que son mucho más flexibles en ese aspecto.
Analizando el error vemos que en el caso de las fechas de expiración, este dato es generado por el servidor externo, por lo que, si no se formateaba de forma correcta, al ser introducido el valor en nuestra base de datos se generaba un error no controlado y el script de interacción con la base de datos PostgreSQL, finalizaba su ejecución.

	id	crawl_id	visit_id	change	creationTime	expiry	is_http_only	is_session	last_acce
1	1	5	5	added	2016-12-29 03	2017-12-29 03:36:18	1	0	2016-12-2
2	2	10	10	added	2016-12-29 03	2016-12-29 05:36:19	0	0	2016-12-2
3	3	10	10	changed	2016-12-29 03	2016-12-29 05:36:19	0	0	2016-12-2
4	4	10	10	added	2016-12-29 03	2016-12-29 03:39:19	0	0	2016-12-2
5	5	10	10	added	2016-12-29 03	2016-12-29 03:39:20	0	0	2016-12-2
6	6	10	10	changed	2016-12-29 03	2016-12-29 03:39:20	0	0	2016-12-2
7	7	10	10	changed	2016-12-29 03	2016-12-29 03:39:20	0	0	2016-12-2
8	8	10	10	changed	2016-12-29 03	2016-12-29 03:39:20	0	0	2016-12-2
9	9	10	10	changed	2016-12-29 03	2016-12-29 03:39:20	0	0	2016-12-2

Figura 3 Ejemplo del formato de fechas correcto

Este error se da en un porcentaje muy reducido y por lo que hemos podido ver en los resultados obtenidos, de manera aproximada este error en el formato de la fecha se encuentra en alrededor de un sitio por cada treinta mil sitios analizados. Dada su baja frecuencia su detección fue más complicada, al fragmentar nuestro test de quinientos mil sitios web en grupos de cinco mil, un simple error provocaba que los datos de un grupo entero no se añadieran a la base de datos y quedaran sobrescritos. Se contemplaron dos soluciones, la no adición de las cookies con campos erróneos o de algún modo intentar solventar ese error. Después de realizar varios análisis vimos que este error siempre se producía en la fecha de expiración, con lo que para nosotros modificar ese valor y colocar una fecha por defecto correcta no suponía ningún inconveniente en el tipo de resultados deseados, pero sí que nos permitía tener acceso a esa cookie para ser analizada posteriormente. De tal modo que se añadió la funcionalidad de comprobación del formato de las fechas haciendo un parsing del formato y en caso de que no tengamos match con el formato correcto se cambia ese reevalúa ese campo con un valor por defecto.

Otro problema al que tuvimos que hacer frente fue la posibilidad de realizar múltiples test en multithreading, utilizando varios hilos de ejecución con el objetivo de alcanzar la meta del análisis de 500.000 sitios de manera mensual por lo que era imprescindible cumplir esa meta en menos de 30 días y dejando un margen para procesar los datos.

Con tal objetivo se pasó a rediseñar la ejecución del crawler ya que de otro modo no éramos capaces de realizar un análisis tal magnitud en el periodo establecido ya que

según los tiempos observados en los análisis conllevaría un período cercano a los 3 meses para realizar la totalidad del test.

Browsers	*CPU (16 Threads)*	*RAM (16000 MB)*	*Elapsed Time*	*Time/site*	*5000 analysis (hours)*	*webs/h*	*500K time(s)*	*500K time(days)*
1	9%	700	1630	16.3	22.63888889	220.8588957	8150000	94.3287037
5	30.00%	3500	430	4.3	5.972222222	837.209302	2150000	24.8842593
10	60.00%	7500	275	2.75	3.819444444	1309.09091	1375000	15.9143519
12	75%	7500	240	2.4	3.333333333	1500	1200000	13.8888889
14	80.00%	7500	219	2.19	3.041666667	1643.83562	1095000	12.6736111
20	100.00%	9000	274	2.74	3.805555556	1313.86861	1370000	15.8564815

Tabla 2 Optimización de ejecuciones en paralelo

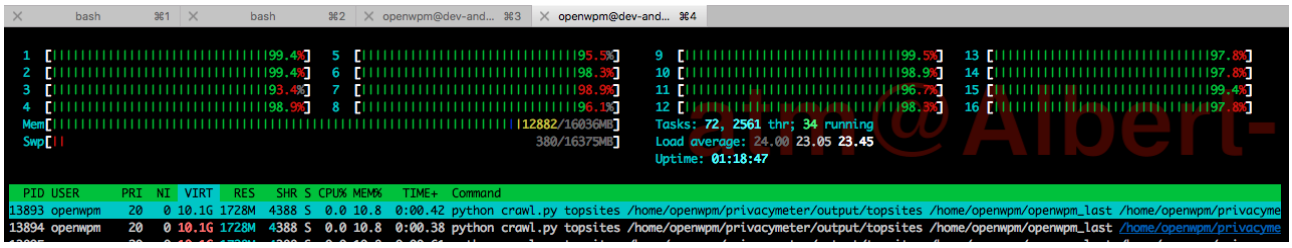


Figura 4 Estado de la máquina durante un test

Los tiempos de la tabla se han calculado de manera precisa generando archivos de log que notifican de la duración exacta en segundos de cada test realizado, se han tenido en cuenta desde el tiempo inicial hasta el post procesado. Este archivo de log generado mantiene un registro de todos los test generados y nos permite estimar los tiempos necesarios y analizar y auditar los errores de manera más efectiva, este archivo de log es independiente de OpenWPM y se ha añadido a nuestra herramienta.

Como vemos, a pesar de que el servidor es capaz de soportar hasta 20 navegadores concurrentes, se decidió en 12 de tal modo que el servidor no quedara inservible y pudiéramos seguir accediendo mediante SSH a él sin problemas (ya que el servidor se encuentra en el distrito Telefónica de Madrid), siendo 14 el valor óptimo si no quisiéramos realizar ningún otro tipo de tarea interna. Para tal uso, se crean un pool de browsers identificados con un id que son pasados a OpenWPM de tal modo que este tendrá disponibles los navegadores y posteriormente vemos como se ejecuta toda la secuencia en monothread en el TaskManager de OpenWPM para evitar posibles errores. El TaskManager se ha modificado de tal modo que cuando se usen múltiples instancias de Firefox, no analicen todo el mismo sitio web (modo interesante para ver lo que sucede al ejecutar diferentes configuraciones como el uso de diferentes pluguins o resoluciones de pantalla), sino que de modo asíncrono cada navegador al quedar libre, analice la siguiente web aún no analizada de la lista.

4.3 Implementación de la herramienta

El objetivo de la herramienta, era hacer uso del crawler OpenWPM de manera totalmente transparente de tal modo que las dos herramientas crezcan de manera

independiente y no realizar un fork de la herramienta proporcionada por Princeton y mantenida a nivel global por su comunidad. Las razones para ello son las siguientes:

- Mantener OpenWPM siempre actualizado con las últimas novedades incorporadas por la comunidad de tal modo que nuestra herramienta sea totalmente compatible con ella.

- Focalizar el trabajo en el desarrollo de Web Privacymeter de tal manera que independientemente de OpenWPM se puede seguir mejorando la herramienta de manera autónoma, referenciando a OpenWPM como una dependencia externa mediante un path hacia el directorio del framework al ejecutar nuestra herramienta.

Durante el transcurso del desarrollo vimos que podíamos aportar mejoras a OpenWPM y se han ido añadiendo, estás mejoras son se explican en el siguiente punto y se han ido añadiendo al repositorio original a medida que se han generado soluciones estables de tal modo que podemos beneficiar a la comunidad de nuestras mejoras.

La herramienta ha ido creciendo a medida que el proyecto ha ido evolucionando. Partiendo de la realización del script donde se llama a OpenWPM hasta la infraestructura final.

En el diagrama de la siguiente imagen se puede apreciar una visión general del ciclo de funcionamiento de la herramienta de generación y almacenamiento de datos. Como podemos ver, se ha creado un hilo de ejecución principal que ejecuta los diferentes módulos de la herramienta desarrollada. Cómo se comentó previamente se puede apreciar el código se ha construido básicamente en Python por su gran flexibilidad e integración con otras herramientas y bases de datos y el conector de los diferentes módulos se ha desarrollado en lenguaje bash ya que era la forma más limpia y robusta para conseguir la funcionalidad deseada.

En la imagen podemos ver la ejecución de un test completo, desde las llamadas a los diferentes módulos de generación de listas, obtención de datos de los sitios web, la generación de la base de datos de los resultados con tal información y preparada para posteriormente añadir el tipo de técnicas de rastreo que se desean, en nuestro caso las técnicas de fingerprinting, pero preparada para posteriormente añadir cookie Syncing y cookie Respawning entre otras posibilidades. La adición de nuevos sitios al test, ya sea desde la web, el pluguin o directamente desde la API.

También se puede observar la interacción con OpenWPM, la creación de la base de datos SQL provenientes del crawl. El posterior análisis de esos datos e incorporación a la base de datos de resultados. La fase de generación de estadísticas en función de los resultados y los datos recopilados asociados a cada sito y su posterior subida a la base de datos Firebase.

En este esquema no se incluye ninguna de las interacciones por parte del cliente, ni web, ni mediante la API ni usando el browser. Esta parte del proyecto, se detallará en el punto de 3.8 dónde se entra en detalle en el proceso de front-end.

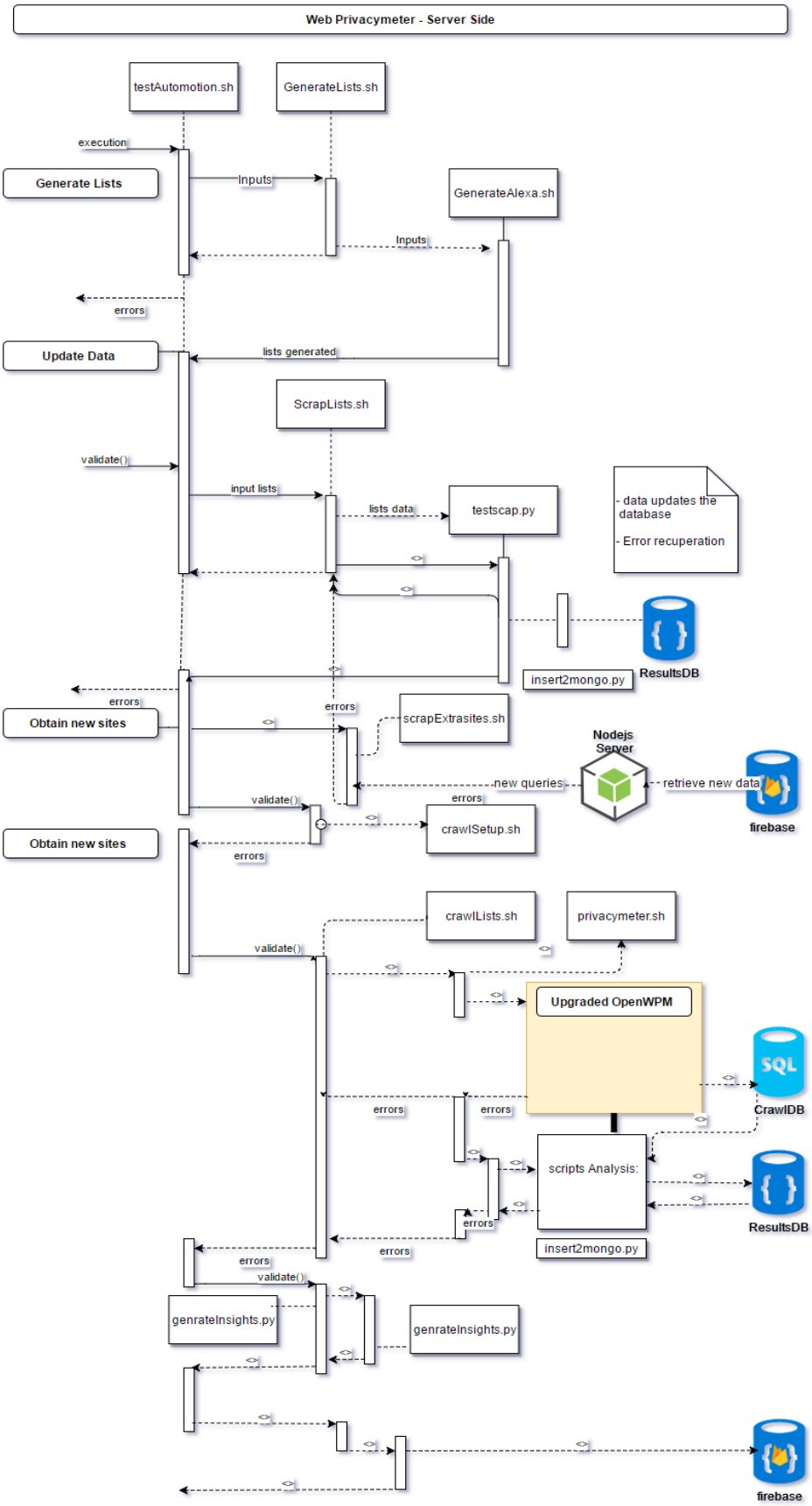


Figura 5 Interacciones herramienta backend

La idea detrás de esta construcción modular es la fácil adición a posteriori de nuevos módulos, la modificación de los mismos de manera independiente y sin influir en el resto y la posibilidad de ejecutar los módulos de manera independiente siempre de manera muy clara y documentada desde la consola de nuestra maquina Linux.

Por ejemplo, si únicamente queremos realizar el Crawling con listas que ya tenemos previamente generadas con una llamada al script privacymeter.sh podemos realizar el test sin problemas: `OPENWPM_PATH=/path/to/openwpm ./privacymeter.sh experiment1 true websites.txt path/to/configuration`

Dónde: `privacymeter.sh <experiment_name> <output_folder> <clean_experiment> <urlfile> <config_folder>'`

Es interesante apreciar cómo se genera una variable global con el path del framework OpenWPM con lo que podemos realmente apreciar que no hay ninguna dependencia entre ambos y son completamente independientes, que como comentado previamente nos permitirían una actualización de cualquiera de los dos sin influir en el otro.

Analizando el repositorio de Privacymeter, podemos ver como se fragmenta el código en seis módulos diferentes y los scripts principales del directorio principal. Los cuáles tienen tareas independientes. Estos módulos son los siguientes.

```
privacymeter
├──analysis
├──configuration
├──conversion
├──export
├──output
├──testAutomotion
└──utils
```

El módulo de análisis es dónde se encuentran todos los scripts para realizar análisis de la base de los resultados provenientes del Crawling y de inserción de los resultados en la base de datos mongodb.

El módulo de configuración es muy interesante ya que nos permite solventar varias problemáticas ya que nos permite configurar el acceso a la base de datos con los parámetros de acceso detallados. De esta manera se resuelve una problemática inicial que requería permisos de superusuario para ejecutar el proyecto y que suponía un problema si se querían realizar test múltiples. También podemos observar que se encuentra la configuración del servicio de correo. Con este servicio se reciben alertas sobre el proceso de ejecución, una característica que es muy importante al realizar test de semejantes duraciones.

El módulo conversion nos permite convertir la base de datos inicial MySQL a formato PostgreSQL, que es con la que trabajamos en los scripts de análisis. Esta conversión es únicamente para añadir simplicidad y velocidad al proceso de creación de vistas.

En export encontramos el código con referencia a la exportación de los datos tanto a la base de datos mongo como a Firebase. También encontramos el código de recopilación de las queries introducidas desde la web o la API a la base de datos de Firebase.

En output encontramos las bases de datos PostgreSQL obtenida de cada test y les resultados de los análisis en formato de texto. Para cada test realizado se genera un directorio con todo el contenido de tal test.

En testAutomotion, encontramos las llamadas a los diferentes módulos y es la secuencia de un test completo. Cada uno de los módulos puede ser ejecutado de manera independiente y desde el script testAutomotion.sh podemos activar o desactivar cada uno de los módulos para que se ejecuten o no durante la cadena.

El objetivo del directorio utils, es englobar el conjunto de herramientas necesarias para determinadas partes del proyecto, encontramos todo el código relacionado con la obtención de información con la que combinar los resultados del Crawling para de este modo, poder sacar conclusiones más interesantes como por ejemplo el número total de usuarios que son rastreados mediante estas técnicas haciendo uso de los tráficos obtenidos mediante webs de análisis. También encontramos en este directorio el servidor nodejs, encargado de analizar si hay nuevas queries en el servidor y en caso que así sea, añadirlas a la base de datos para ser añadidas posteriormente a nuestro test.

4.4 Scripts de análisis

Los scripts de análisis son cruciales en el desarrollo de este proyecto y el motivo que lo hace tan escalable e iterable a largo plazo analizando, el motivo es que una vez realizado el Crawling, todos los datos se encuentran en la base de datos y como consiguiente, pueden ser manipulados independientemente del crawl. Se decide realizar scripts de análisis de tal modo que las consultas a la base de datos sean automáticas y se puedan obtener los resultados deseados sin necesidad de analizar las bases de datos manualmente, de tal modo se puede automatizar y dejar la herramienta analizando la base de datos en búsqueda de los resultados deseados. Para esta primera parte del proyecto se han realizado cuatro scripts de análisis que como el resto del código se adjuntan cómo anexo.

Para detectar las técnicas de fingerprinting no es tan fácil como ver si se están usando las APIs usadas para tal fin o los métodos en concreto de tales APIs ya que estas APIs son usadas con muchos objetivos que nada tienen que ver con el fingerprinting. El método que tenemos para detectar el fingerprinting es la secuencia en que toman parte tales invocaciones de tal modo que no dan lugar a duda de la intencionalidad de generar un fingerprint. Cada tipo de fingerprint tiene su secuencia temporal de llamadas a las APIs en función de la acción a realizar. En los siguientes apartados se van a detallar estas secuencias según el tipo de fingerprint.

Estos scripts pueden ser ejecutados independientemente analizando el contenido de una base de datos:

```
python script_analysis.py <dbname> <results_dir> <configfolder>
```

Posteriormente, analizaremos cada uno de los scripts de análisis dependiendo del objetivo a realizar, ya sea para detectar tipos de fingerprinting o minado de cryptomonedas.

4.5 Complementación de resultados con datos externos

Durante el desarrollo de este proyecto siempre se ha considerado como un pilar del proyecto la difusión del contenido de una manera que llegue, hasta ahora test de valoración de privacidad únicamente tenían en cuenta la posición en el ranking global de Alexa. Bajo el punto de vista del departamento era imprescindible dar más información que hiciera esos resultados mucho más intuitivos y permitiera estudiar más en detalle posibles vínculos que se pudieran establecer.

La información que se ha requerido durante el proyecto ha variado. Inicialmente se decidió optar por el análisis del top 500 de cada país y cada categoría que nos proporciona Alexa (Alexa, 2017), analizando un total de 75000 sitios web.

Para poder obtener las listas de manera gratuita se implementó el primer script de scraping. El termino web scraping se usa para hacer referencia al uso de un programa para almacenar y procesar el contenido de un sitio web. En nuestro caso, para acceder a las listas sin hacer uso de la API que ofrece Alexa que es de pago, hacemos un scrap automatizado de las listas. Para realizar el scraping es necesario determinar cómo está estructurada la información para posteriormente recopilarla. Posteriormente se ha realizado en Python principalmente usando la librería urllib2 que nos permite abrir una url http y obtener su contenido. Posteriormente, una vez tengamos el contenido HTML de la página pasamos a analizarlo.
Google.es

Podemos observar como estos links van a seguir siempre el mismo formato por lo que podemos de algún modo detectarlos. Esa detección se ha decidido realizar partiendo el contenido por el carácter ‘<’ de tal manera que obtenemos una lista con el inicio de cada tag. Una vez tengamos separados todos los tags, únicamente debemos filtrar los que contienen la estructura correspondiente a un sitio, para ello se mira si los elementos de la lista empiezan por la secuencia a href="/siteinfo/, si es así significa que hemos detectado un nuevo sitio web para añadir a las listas, lo formateamos correctamente y se añade. Este test se realiza 20 veces para conseguir el top 500 ya que Alexa indexa 20 sitios por página.

Después de trabajar un tiempo vimos que estos rankings de Alexa nos dan la información de las webs con más visitas por cada país, pero no por ello de ese país. Esto es un problema ya que vemos que sitios web como Google.com o Facebook.com aparecen en las listas de todos los países por lo que no estábamos maximizando la obtención de sitios webs útiles ni teníamos un censo tan grande como nos gustaría.

Con el objetivo de realizar un test muy superior en número e información asociada, se exploran nuevas formas de obtener esos datos. Se llega a la conclusión que la opción

más viable en cuanto a coste es coger la lista pública de Alexa del millón de webs más visitadas. Al perder la información sobre categorías o países, tenemos la necesidad de incorporar alguna nueva forma que nos ofrezca la información de los sitios web.

En esta búsqueda, se decide que la mejor opción por precisión es el uso de la herramienta Similarweb catalogada como la opción más precisa en el cómputo de los tráficos (Rand, 2015). En el estudio se compara la información estimada por los servicios con la real proporcionada por el administrador del sitio web ver a continuación, vemos que el coeficiente de correlación de Spearman en SimilarWeb es superior al de Alexa.

SimilarWeb:

Spearman's Correlation w/ Actual Traffic: **0.839**

Standard Error: 0.0435

Data coverage: 97.2%

Alexa:

Spearman's Correlation w/ Actual Traffic: **0.702**

Standard Error: 0.0607

Data coverage: 100%

Con el objetivo de complementar la información de los sitios web con más información se realiza un nuevo scrap ya que al igual que Alexa la API es de pago y se ha intentado minimizar el coste del proyecto, esta vez del servicio que ofrece SimilarWeb. Este scrap es más complicado que el de Alexa ya que la información no está formateada del mismo modo al no ser una lista. Para ello usaremos las librerías de Python requests y html de lxml. Con la librería requests, descargaremos el contenido del sitio y posteriormente con la librería html parsearemos su contenido en formato html para posteriormente analizar su contenido:

```
page = requests.get('https://www.similarweb.com/website/' + url, verify=False)
tree = html.fromstring(page.content)
```

Posteriormente, la variable tree contendrá todo el contenido del html, y usaremos la herramienta XPath para localizar los campos del sitio web que queramos.
En el ejemplo de la imagen, vemos que el contenido que en este caso nos interesa, el ranking del país, se encuentra en el elemento rankingItem-value, con XPath vamos a realizar una query al HTML almacenado para obtener su valor:

```
ranks = tree.xpath("//span[contains(concat(' ',normalize-space(@class),' '),' rankingItem-value ')]/text()");
```

Figura 6 Ejemplo de scrap de resultados

Posteriormente, se ha generado todo el código de control de errores y de almacenamiento de los datos.

4.6 Definición de las estructuras de datos

La definición de las estructuras de datos siempre ha ido enfocada a la escalabilidad, de tal modo que, si nuevos test son añadidos para su análisis, podamos incorporar esa información de manera directa sin tener que realizar cambios en la base de datos.

Por ese motivo, para almacenar los resultados, a diferencia de la información recopilada mediante el crawl, se ha decidido almacenar los resultados en una base de datos NoSQL en concreto MongoDB. El motivo de su uso tiene varios motivos, las bases de datos no relacionales como MongoDB, a diferencia de las bases de datos relacionales, la información se almacena en documentos a los que accedemos con una llave. Nos es muy útil ya que permite tener mucha flexibilidad ya que no todos los documentos tienen porque tener los mismos campos como nos ocurre en nuestro caso. Nos permite hacer uso de campos anidados, opción que nos es muy interesante para almacenar la información.

Teniendo en mente esa estructura se han definido las diferentes colecciones de la base de datos mongoDB y también la estructura de sus documentos. La base de datos se ha divido en cuatro colecciones principales:

Sites: Encontramos todas las webs analizadas resultados obtenidos y su información adicional. Un ejemplo en formato JSON sería el siguiente:

```
“7a616d65656e2e636f6d”: {
          "fingerprint": {
                    "201612": {
                              "canvas": [
                                        "http://zameen.com/ga094003.js"
                              ]
                    }
          },
          "information": {
                    "audience": [
                              "Business and Industry > Real Estate",
                              "Internet and Telecom > Chats and Forums",
                              "News and Media",
                              "Shopping > Classifieds"
                    ],
```

```
                    "category": "Business and Industry > Real Estate",
                    "country": "Pakistan",
                    "country_rank": "116",
                    "global_rank": "11449",
                    "traffic": "1300000",
                    "traffic_top": {
                              "Pakistan": "0.8009",
                              "Saudi Arabia": "0.0391",
                              "United Arab Emirates": "0.0474",
                              "United Kingdom": "0.0149",
                              "United States": "0.0218"
                    }
          },
          "url": "zameen.com"
}
```


Scripts: listas de scripts detectados e información relacionada con ellos como el número de veces que aparece cada uno o el dominio al que pertenece en caso que sea posible ya que muchas veces no se puede automatizar ya que estos scripts se distribuyen mediante CDNs lo que imposibilita encontrar el dominio de manera directa.

Insights: En esta colección se añaden todos los cálculos realizados que nos permiten tener una visión general de los que está sucediendo. Datos como el número de webs que rastrean por categoría o por país, el número de usuarios que son rastreados mensualmente o los históricos que nos van a permitir ver las tendencias que siguen estas las prácticas analizadas en el transcurso de los diferentes test analizados.

Queries: Las webs que los usuarios consultan y no se encuentran en nuestra base de datos, se almacenan en esta lista para posteriormente con el servidor nodejs que escucha cuando una nueva query es añadida y la manda al servidor para ser procesada y realizar el análisis para posteriormente subirla al a base de datos para que el usuario pueda consultar el resultado.

Hay que tener en cuenta que la estructura usada debe ser compatible con la que vamos a usar en Firebase, por lo que vemos que el que a priori parece que debería ser el identificador de cada sitio web, su url, no puede ser usado ya que el punto "." Es un carácter usado de manera interna por Firebase y no es posible usarlo como llave. Debido a esta limitación se nos plantean dos posibilidades. o usar secuencias de scape lo que significaría decidir un carácter de substitución y tener siempre en cuenta esa decisión o codificar el string con la URL. Para este proyecto se ha decidido implementar esta segunda opción y codificar los caracteres del string en base16. Es una opción muy interesante ya que nos resuelve cualquier problemática que pueda surgir y además con mucho soporte tanto en el backend realizado como en el frontend en JavaScript.

4.7 Generación de resultados globales: "insights"

Estos scripts tienen el objetivo de iterar sobre los datos obtenidos para cada sitio y generar resultados que dentro del proyecto se les ha llamado insights. Este análisis se ha realizado con los datos que encontramos en MongoDB posterior al análisis del crawl.

Este análisis se ha ejecutado mediante scripts Python, para ello se ha usado la librería pymongo, imprescindible para la interacción que necesitamos con la base de datos. Los datos de interés son varios:

- Países que tienen el hosting de más sitios usando fingerprint.
- Top países con más tráfico rastreado usando técnicas de fingerprint.
- Categorías de sitio web que más usan esas técnicas de fingerprinting.
- Categorías con más tráfico rastreado con técnicas de fingerprinting.
- Estimación del tráfico total rastreado mediante las técnicas analizadas.
- Histórico de la evolución del uso de estas técnicas.
- Porcentajes de uso de técnicas de fingerprinting respecto al total.

Para obtener estos valores, se ha iterado sobre los resultados, gracias a las posibilidades que nos ofrece MongoDB y la adaptación de esta al proyecto y combinado con pymongo hemos podido reducir el nombre de queries necesarias.

Para agilizar el proceso, se ha introducido el flag is_tracking en la base de datos, por defecto False que en cuanto se detecta un script de fingerprinting se cambia a True, por lo que posteriormente podemos filtrar por ese flag mediante pymongo o directamente desde la consola de MongoDB para obtener únicamente las entradas con ese flag a True.

Un ejemplo de ejecución del código comentado sería el siguiente:

```
client = MongoClient('localhost', 27017)
       db = client[dbName]
       collection = db[collection]
       for doc in collection.find({"is_tracking": True}, {"_id" : 0, "is_tracking":
0}):
               #Code to be executed
```

Como vemos en el código se filtra por el parámetro la query de búsqueda y posteriormente se notifican los parámetros que deben o no mostrarse, como vemos al estar marcados como 0 no se mostrarán el output de la query. Posteriormente se va a iterar sobre todos los documentos resultantes de tal modo que accediendo a lso datos podemos agrupar sus propiedades y contando las repeticiones generar las diferentes estadísticas deseadas. La query variara en función de las estadísticas deseadas.

También se generan listas de scripts de tal modo que el usuario va a poder acceder a esas listas de tal modo que van a poder ser usadas por usuarios finales o desarrolladores que las utilicen en herramientas de privacidad como adBlock o Ghostery. Durante la conferencia en Columbia se establecieron contactos con la empresa RedMorph, centrada en la privacidad, que también mostro gran interés en el uso de estos resultados para su utilización en sus productos.

5. Análisis de la privacidad en la web

Para comprender las técnicas de fingerprinting, cookie syncing y de rastreo en general, hay que remontarse a las necesidades iniciales de la industria y a los cambios que se han ido produciendo.

La privacidad de los usuarios en la red ha ido cambiando a medida que la web ha ido evolucionando.

Inicialmente los sitios web eran "stateless", es decir, no tenían memoria (estado) donde se pudiera almacenar información sobre el usuario.
En 1994, se introdujeron las webs "stateful", utilizando el HTTP State Management Mechanism. En síntesis, utilizando este mecanismo, las webs pasan a tener memoria donde se mantiene información/estado del usuario. Las webs "stateful" se basan en la utilización de cookies.

Desarrolladas por Lou Montulli, las cookies son archivos de tamaño reducido generados por el servidor y que después de ser enviados mediante la conexión HTTP, se almacenan en el cliente. Este a su vez, cuando se conecta a un servidor del cual contiene cookies previas, las transmite en el encabezado del request HTTP.

Por defecto, la mayoría de navegadores tiene las cookies habilitadas. Cuando el usuario accede a un sitio web por primera vez, el servidor envía información de la sesión y esta se almacena en el ordenador del usuario. Simultáneamente, el servidor almacena esta misma información. Cuando un usuario accede a un servidor web, este compara los archivos del lado del servidor y los archivos del lado del usuario. Este mecanismo permite a los sitios y aplicaciones web almacenar datos en los dispositivos de los usuarios para su uso posterior.

Las cookies son enormemente beneficiosas y facilitan la interacción del usuario con la página web, en situaciones donde el usuario tenga que autentificarse o e-Commerce.
Pero también pueden comprometer, y de hecho comprometen, la privacidad del usuario, cuando se utilizan para la realización de perfiles de usuario que se utilizarán por proveedores de publicidad u otros más peligrosos.

Las cookies acostumbran a diferenciarse entre dos grandes tipos no según su estructura, ya que son exactamente iguales desde el punto de vista técnico sino por quién las está generando e insertando en una página web. y que al fin y al cabo acostumbra a implicar un uso completamente distinto.

El primero, las "first party" cookies, son las que referencian a cookies generadas por el mismo sitio web dónde nos encontramos y que normalmente tienen propósitos lícitos como mantener una sesión iniciada o por ejemplo en e-Commerce recordar la cesta de la compra para la próxima vez que accedamos al sitio web.

Por otra parte, las "third party" cookies, acostumbran a tener objetivos completamente distintos y comprometedores para la privacidad de los usuarios. El funcionamiento de este tipo de cookies es exactamente el mismo con la diferencia que estas cookies no

están siendo generadas por el sitio web al que accedemos, sino por recursos remotos a los que un usuario accede para conseguir recursos que no se encuentran alojados en el servidor del sitio web, normalmente anuncios y añade una cookie en caso de que no se haya añadido previamente. El siguiente esquema ayuda a entender el funcionamiento de esta técnica.
El algoritmo de funcionamiento de las "third party" cookies es el siguiente:

1.El usuario solicita un sitio web
2.El sitio web envía el contenido del sito y en tiempo de ejecución se solicitan los anuncios/banners al proveedor.

3.Se solicita al proveedor de anuncios el contenido.

4.El distribuidor responde con el anuncio juntamente con una cookie.

5.Cada vez que el usuario requiera información del anunciante, le devolverá la cookie con lo que se identificará y podrá ser rastreado.

Esta práctica permite a las "third party", rastrear a los usuarios ya que esa cookie generada por una "third party" será devuelta a él en todos los sitios dónde aparezcan recursos de su dominio. Generando un mapeado de todos los sitios web visitados y al fin y al cabo rastreando el usuario por la red. Estas técnicas que mayormente son usadas para definir los intereses de un usuario y poder mostrar anuncios en consonancia, generan una gran controversia debido a la perdida de privacidad que esta metodología implicaba.

La incorporación automática de banners publicitarios, etc de terceros, deja al propietario de la página web, a oscuras del uso que estos proveedores están haciendo de las cookies en su propia página web, y del riesgo al que está sometiendo, de manera involuntaria, a los visitantes de su sitio.

Las cookies, al estar perfectamente definidas tanto ellas como la forma de interactuar con ellas por la Internet Engineering Task Force (IETF) (RFC6265 Barth, 2011), son fácilmente manipulables y, aunque pueden ser generadas para perdurar en el tiempo, pueden ser borradas fácilmente en el navegador, o incluso bloqueadas mediante extensiones u opciones del mismo.
En ese caso, el perfil generado hasta ese momento se desvincula del usuario y hay que volver a empezar a generar un perfil nuevo con la consiguiente pérdida de información y su valor.
Por tanto, desafortunadamente para los "trackers" la decisión de permitir ese rastreo mediante cookies reside en gran medida en el usuario. Lo mismo sucede con otras técnicas como las "Flash cookies" o los "entity tags" que han sido usados de la misma manera como sustitutivo o como sistema de respaldo de las cookies tradicionales, con las mismas limitaciones.
Por este motivo, los esfuerzos de los "trackers", principalmente relacionados con la industria de la publicidad, han concentrado sus esfuerzos hacia el desarrollo de

diferentes técnicas, que sean difícilmente bloqueables, y que permitan mejorar la trazabilidad de los usuarios.

Estas técnicas son las que se detallan en los próximos puntos. La detección de los sitios web que las utilizan se ha llevado cabo en el curso de este proyecto.

5.1 Estado del arte

De la misma manera que la huella dactilar identifica de manera única a una persona, utilizando una característica individual, las técnicas de "fingerprinting" se utilizan para conseguir un identificador único e invariante del equipo que utiliza el usuario.

Browser "fingerprinting" es una técnica de rastreo en navegadores web cuyo uso se está generalizando, para identificar un usuario individual usando multitud de parámetros e información visible por el navegador cuando se visita una página.

Estos identificadores pueden aparentar ser genéricos y no identificar de manera individual a un usuario. Sin embargo, en la práctica, la utilización de múltiples métodos fragmentados y simultáneos permite identificar a un usuario con una alta probabilidad.

Los parámetros recopilados se envían al servidor, que genera un identificador único propio y global del dispositivo, y precisamente, el guardar este identificador único y a priori inmutable del dispositivo, es lo que lo diferencia de otras técnicas de rastreo como el uso las direcciones IP o las cookies, ya que las técnicas de "fingerprinting" no son volátiles y pueden persistir de manera indefinida en el tiempo.

Estos identificadores únicos constituyen un alto riesgo para la privacidad de los usuarios. Es mucho más complicado "mutarlo" o interrumpir el rastro que un usuario está dejando por la red.

Las técnicas de "fingerprinting" pueden usarse de manera independiente o combinadas con otras técnicas para de ese modo reforzar su efectividad creando perfiles de usuario. Una práctica habitual es la generación fingerprints al mismo tiempo que se generan cookies. De esta manera si en un momento dado se eliminan las cookies y se generan nuevas con la consiguiente pérdida de información que ello implicaría, mediante el uso de "fingerprinting" los dos historiales de cookies pueden vincularse y seguir formando un histórico del usuario a pesar de explícitamente haber eliminado las cookies.

Estos métodos son opacos y no únicamente son imperceptibles para los usuarios, sino que herramientas como Ghostery o Adblock no son capaces de detectar.
Únicamente es posible bloquear o advertir al usuario en caso de que este intente acceder a una página que las utilice, gracias a estudios como el realizado en este proyecto.

Otros navegadores como TOR, centrados en la privacidad, incorporan herramientas para reducir el fingerprinting, pero aun así no es posible asegurar el bloqueo total.
Por este motivo, es muy interesante tener un test periódico con un gran censo de páginas web, que permitan observar las tendencias de las diversas técnicas utilizadas y combatir las fuentes de manera fiable.

Las técnicas de “fingerprinting” han ido evolucionando a un ritmo muy elevado desde que en 2009 se empezaron a documentar, utilizando métodos que cada vez son más sofisticados.

En este proyecto se analizan las formas más novedosas utilizadas para el browser fingerprinting. La información recopilada puede ser de tipo muy distinto (no está estandarizada) y puede incluir el tipo de navegador y la versión, el Sistema operativo y la versión, la resolución de pantalla, las fuentes soportadas, los plugins, la zona horaria, la configuración de idioma, y un largo etc.

Los distintos tipos de “fingerprinting” al ser, a priori, métodos ortogonales entre sí, pueden ser combinados para generar mayor entropía y unicidad del identificador final. En consonancia con las third party cookies, las técnicas de “fingerprinting” son utilizadas por empresas anunciantes. Siguiendo un algoritmo muy similar y que se describe a continuación.

1.El usuario solicita un sitio web

2.El sitio web envía el contenido del sito y en tiempo de ejecución se solicitan los anuncios al proveedor de publicidad.

3.El distribuidor responde con el anuncio conjuntamente con un script de “fingerprinting”.

4.El script se ejecuta en el cliente, se obtiene su fingerprint y se envía de vuelta al proveedor de publicidad.

5.El proveedor de publicidad analiza ese identificador o confía en un proveedor de “fingerprinting” para finalmente identificar al usuario.

6.El proveedor de publicidad puede identificar al usuario por todos los sitios web dónde aparezcan sus anuncios, sin que propietario de cada uno de esos sitios sea consciente del rastreo a que se somete a sus visitantes.

Es importante incidir en que el factor más importante de un fingerprint es su unicidad. Un fingerprint será útil mientras sea capaz de diferenciar un equipo del resto de dispositivos con la mayor probabilidad posible. Por ese motivo se utilizan de manera simultánea el mayor número posible de distintas técnicas de fingerprinting.

La combinación de ellas genera más entropía en el identificador único final, pudiendo llegarse a identificar a un usuario con una probabilidad del 99.1% según (Eckersley, 2010).

Aunque, aparentemente el “fingerprinting” puede ser menos efectivo en dispositivos móviles de grandes fabricantes que comparten exactamente el mismo hardware, algunos proveedores de librerías de “fingerprinting” como Augur (Augur Technologies

Inc), aseguran que, utilizando información del operador de telefonía en el fingerprint, pueden identificar a un usuario con una probabilidad del 94%.

En los próximos apartados se analizan en detalle el uso de distintas técnicas para generar fingerprints usando HTML5. Se incluyen también las que hasta este proyecto no habían sido analizadas de manera continua.

Este análisis permitirá obtener conclusiones sobre las tendencias de la utilización de las distintas técnicas.

5.2 Canvas fingerprinting

A medida que los navegadores van añadiendo funcionalidades cada vez más complejas, empiezan a utilizar más recursos del sistema operativo, y el comportamiento del navegador es más dependiente de esos recursos, sobre todo desde la incorporación de HTML5 y sus APIs.
Canvas fingerprinting es una de estas APIs y el método de fingerprinting más extendido de los analizados seguidamente.
Esta API se basa en la capacidad de HTML5 de dibujar programáticamente, texto 2D, objetos 2D <canvas>, objetos 3D (WebGL), y examinar los pixels producidos en pantalla.
Examinando los pixeles producidos en distintos escenarios, por ejemplo, dibujando texto sin especificar la fuente o especificando distintas fuentes y distintos tamaños, dibujar superficies 2D y/o animaciones 3D, permite generar un hash identificativo o complementar uno existente añadiendo más entropía.

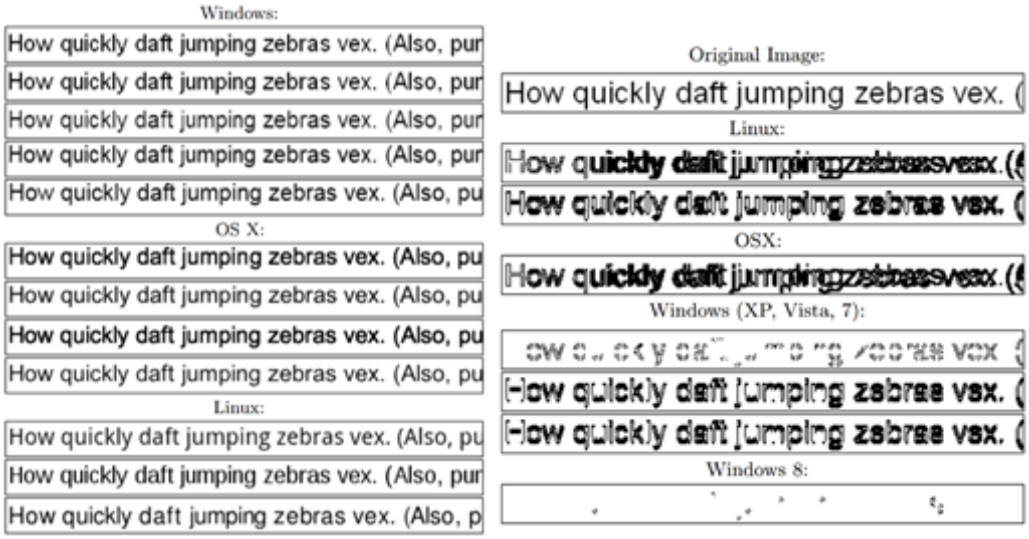


Figura 7 Renderizado de texto en distintas máquinas y distintos SO (derecha) Substracción del texto renderizado y el original (izquierda)

Esta técnica se basa en la premisa de que cada ordenador tiene una configuración distinta que influye en el renderizado de una imagen

(Shacham, 2014), elementos como el sistema operativo, la tarjeta gráfica, la versión del navegador, las fuentes instaladas, el renderizado de subpixels o el antialiasing toman parte en la generación del bit-map final. En la figura de la izquierda, se pueden observar las diferencias en el renderizado de un texto utilizando la fuente Arial en distintos ordenadores y distintos sistemas operativos. En la figura de la derecha se resaltan las diferencias en el renderizado en distintas máquinas y SO sustrayendo la imagen original de la renderizada en cada equipo.

Figura 8 Pequeñas variaciones generan un hash completamente distinto

Como hemos comentado previamente, cualquier fingerprint debe ser generado en el cliente y en este caso en concreto se genera una imagen con el mismo texto y figuras que, según las características del dispositivo, presenta las variaciones.
La página web dibuja los elementos del test en un <canvas>, extrae el mapa de pixeles resultante y genera un hash único y consistente en múltiples ejecuciones de la página en la misma máquina.
Este tipo de fingerprint usa varios métodos de la API canvas de HTML5 para realizar los pasos necesarios de recopilación del hash, su detección no es fácil, ya que no es una única llamada a la API que delata esta intencionalidad de identificación, sino que son varias llamadas en una secuencia determinada que dan lugar a la generación de la imagen y del hash.

Los diferentes métodos utilizados de la API canvas serán comentados en el apartado de desarrollo del proyecto durante la exposición de los scripts de análisis realizados.
El canvas fingerprinting alcanza fácilmente una alta entropía de 5.73 bits o lo que es lo mismo, un factor de unicidad del 0.9807 lo que le convierte en uno de los métodos de fingerprinting stand alone más potentes. [JT2] (Keaton Mowery and Hovav Shacham 2012)

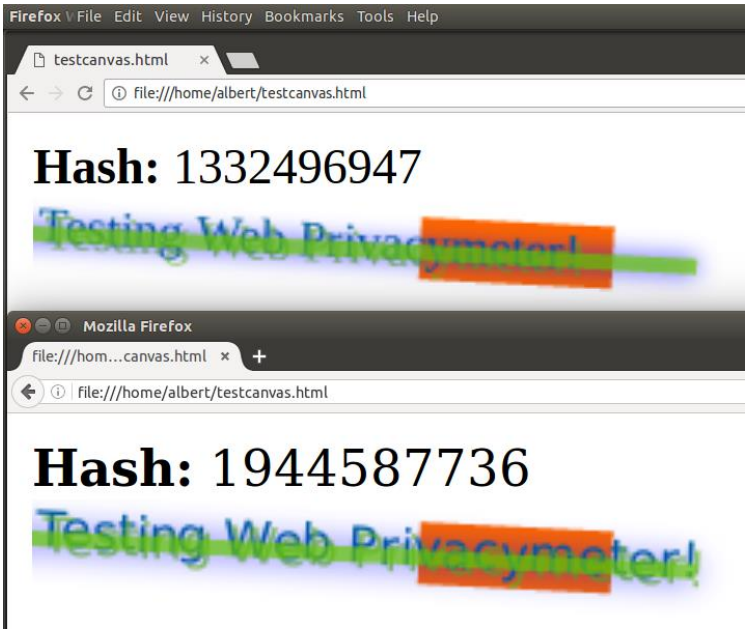


Figura 9 Diferencias del hash y el canvas en función del navegador utilizado Chrome Arriba, Firefox Abajo

En esta última figura podemos ver la imagen que he generado a modo de prueba para el proyecto para generar un fingerprint, vemos como a pesar de ser el mismo dispositivo, únicamente cambiando el navegador hay pequeñas características como las fuentes usadas por defecto o el tipo de sombreado que lo hacen único.

5.3 Font fingerprinting

El font fingerprinting, también conocido como font-canvas fingerprinting es una técnica similar al canvas fingerprinting, que usa su misma API realizar el hash.
Su uso, por otra parte, es diferente.
Este método se basa en medir las dimensiones de los glifos (representación gráfica de los caracteres) de las fuentes del texto, ya que cada navegador y versión los renderiza con sutiles diferencias, también las dimensiones variaran si una fuente está instalada o no, ya que en ese caso se usará la fuente por defecto.

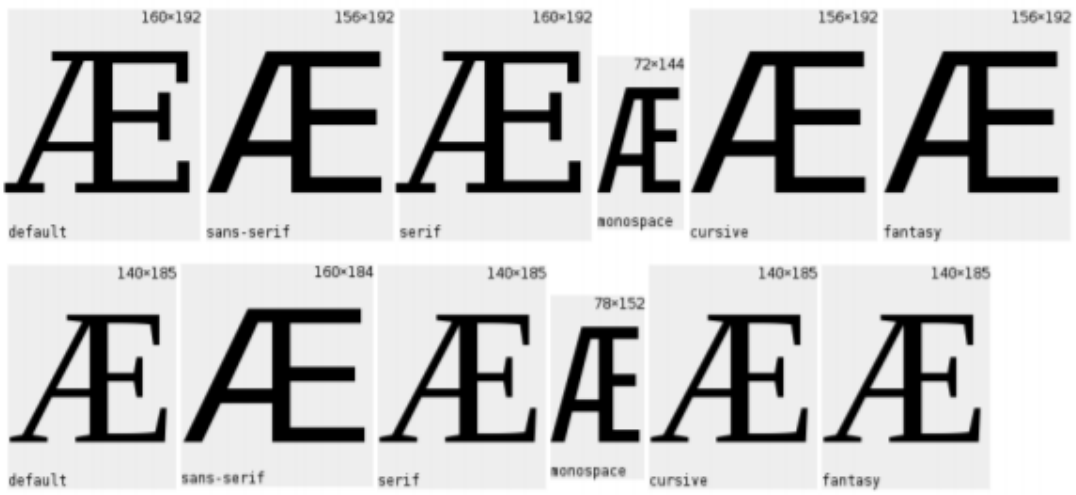


Figura 10 Diferencias al renderizar. Firefox 24(arriba) Chromium 35(abajo).

En esta imagen se constata que los caracteres se renderizan en tamaños diferentes, y que ambos navegadores han elegido diferentes tipografías para las fuentes cursive y fantasy, lo que ayuda a singularizar más los resultados obtenidos.

El Font fingerprinting, tiene por tanto potencial para identificar las fuentes que el navegador tiene instaladas. Si una fuente no estuviese instalada, se utilizará la fuente por defecto y podrá ser detectado al proceder a medir el resultado gráfico del navegador. Cada fuente se renderiza de manera distinta en función del navegador utilizado, lo que nos permite identificarlo.

En la figura 7, se muestran los resultados obtenidos al ejecutar un test de Font fingerprinting. Se muestra la longitud del texto y la representación gráfica del texto. Visualmente podemos observar que el texto que debería utilizar la fuente Roboto tiene la misma apariencia que la fuente por defecto. Ello nos permite inferir que la fuente Roboto no está instalada y que se está utilizando la fuente por defecto en su lugar.

La medida de la longitud del texto que debería utilizar la fuente Roboto, idéntica a la longitud de la fuente por defecto, permite a la API detectar que la fuente Roboto no está instalada. Una singularidad más para generar el identificador fingerprinting único.

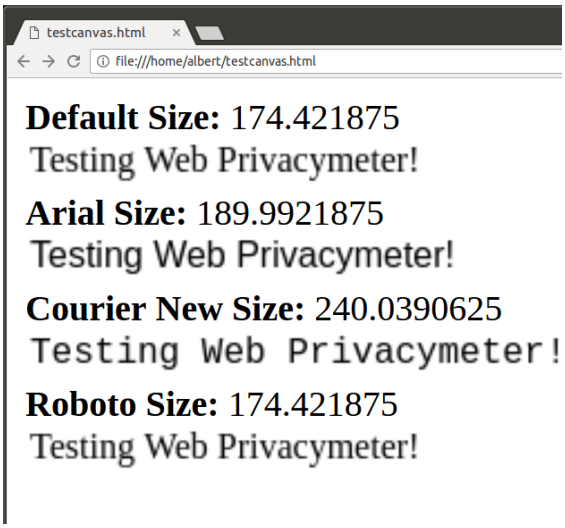


Figura 11 Comparativa longitud de fuentes

Con esta técnica se consigue identificar inequívocamente el 34% de los usuarios (David Fifield and Serge Egelman, Berkeley 2015) y el resto ser subdivididos en grupos más reducidos.

Por otro lado, una propiedad de este tipo de fingerprinting es su ortogonalidad respecto a otros tipos de fingerprinting, lo que la convierte en una técnica complementaria muy útil.

5.4 WebRTC fingerprinting

WebRTC como Canvas, es una API de HTML5 (API webRTC, 2016). Esta API es utilizada para establecer conexiones multimedia P2P entre usuarios, casos habituales serian video-llamadas o streaming de contenido. Para poder establecer esa conexión WebRTC requiere conocer las direcciones origen y destino, por lo que transmite sus direcciones IP privadas (Ethernet o WiFi) sin necesidad de ningún permiso explicito por parte del usuario.
Para descubrir las IP privadas se utiliza la API RTCPeerConnection que es la encargada de establecer las conexiones, tiene acceso a múltiples parámetros y no precisa permisos explícitos por parte del usuario.
La IP pública ya es conocida sin necesidad de fingerprinting.
WebRTC también nos permite conocer los identificadores únicos de los elementos multimedia de cada dispositivo (micrófonos o cámaras ente otros). A pesar de no conocer el modelo concreto esta información nos da un identificador único muy valioso para las prácticas de fingerprinting. En la siguiente imagen se puede observar un ejemplo del uso de la API para realizar fingerprinting (browserleaks, 2016).

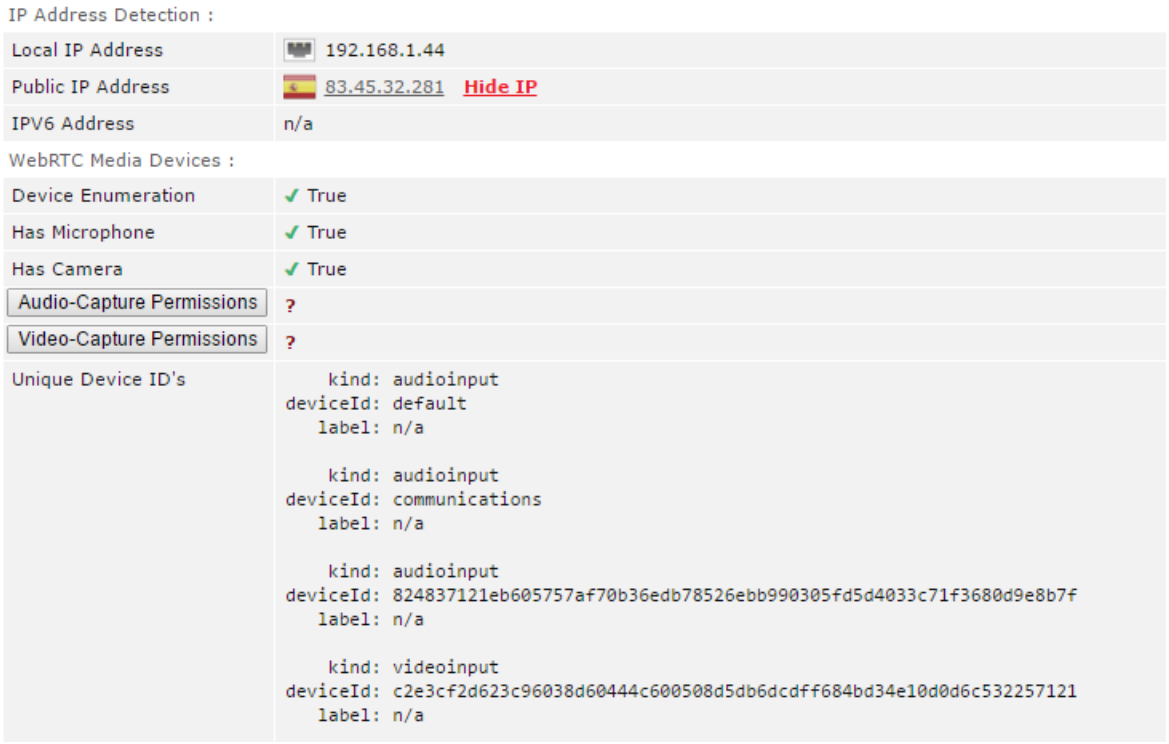


Figura 12 Información recopilada con API WebRTC

5.5 Audio fingerprinting

Esta técnica hace uso de la API HTML5 AudioContext (Mozilla, 2016). Es de las tecnologías estudiadas, sobre la que menos se ha trabajado, y como tal, aún es imprecisa y está bajo investigación. Según el estudio realizado por la universidad de Princeton (Narayanan, 2016), se puede lograr una entropía de 5,4 bits. Al ser aún bastante desconocida, su uso aún es minoritario y es la técnica menos usada del conjunto.

Se han podido observar dos tipos de técnicas, una básica que únicamente comprueba la existencia de las APIs y añade un bit al fingerprint generado por otras de las técnicas. Otra forma de usar AudioContext para rastrear a los usuarios, consiste en una implementación similar a la de canvas fingerprinting.

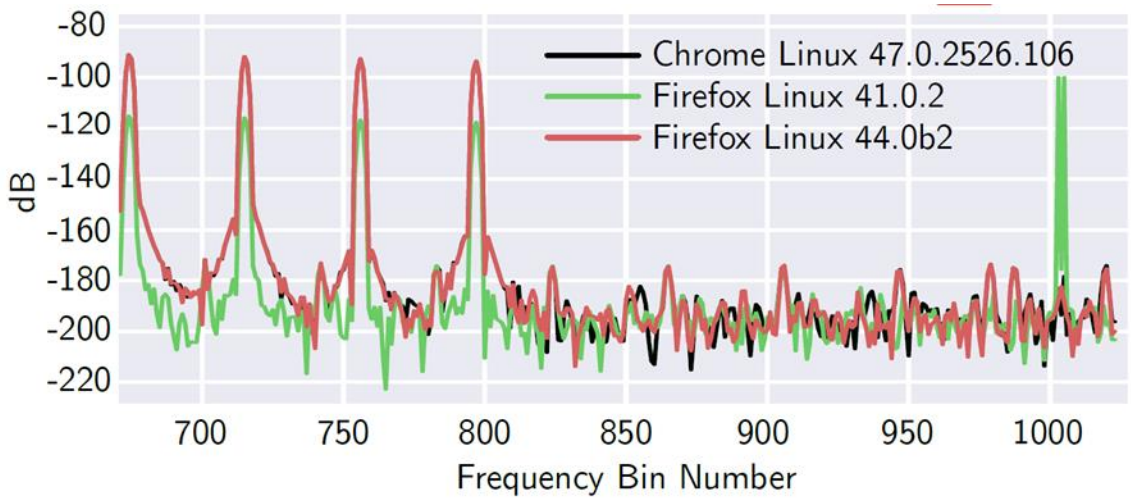


Figura 13 Demostración de audio fingerprinting

En esta imagen se puede observar que las dos señales a priori idénticas tienen diferencias de amplitud e incluso la máquina con Linux 41.0.2 y Firefox entrega una señal con un armónico a frecuencias elevadas que no aparece en los otros casos. Es una demostración más de que distinto hardware produce resultados únicos.
Esta técnica, al igual que canvas fingerprinting no necesita dibujar la imagen, el Audio fingerprinting no necesita reproducir el audio ya que únicamente se procesa la señal sin llegar a mandarse al altavoz.

5.6 Cookie Syncing

Cookie Syncing es otra técnica que se ha estudiado durante el desarrollo del proyecto, sus scripts de detección están en proceso de desarrollo actualmente.
Esta técnica se basa en las cookies por lo que es complementaria al fingerprinting como ya hemos visto anteriormente. Esta técnica se basa en que distintos "trackers" comparten las cookies, ampliando la base de datos individual de cada tracker a una base de datos conjunta. El proceso empieza cuando un usuario visita un sitio web con, por ejemplo, un rastreador third-party A. Esta empresa rastreadora A, recopila las cookies que ese usuario contiene y extrae su identificador, es entonces cuando esta empresa redirecciona al usuario a la empresa B con la que se compartirán los datos, esta redirección se hace pasando el identificador del usuario utilizado por A como parámetro en la url (ej. B.com?A.com=12345 dónde 12345 es el identificador del usuario utilizado por la empresa A). La empresa B recibe las cookies del usuario y además el parámetro generado por A con lo que la empresa B tiene el identificador que utiliza A para identificar el usuario (ej. 1234) y el identificador usado por ella misma (ej. 5678) para identificar al usuario. Este usuario queda mapeado por ambas empresas de tal manera que la empresa B → usuario 5678 es conocido en A como 1234. Posteriormente ambas empresas pueden intercambiar de forma ajena a la navegación, información del usuario que beneficia a ambos trackers.

La siguiente imagen representa este fenómeno de unión de bases de datos dónde se estudia la información de un usuario, como vemos los dos trackers independientes tienen parte del historial de sitios web que se visitan ya que tienen sus cookies en ellas, al realizar la unión vemos que ambos de benefician de una base de datos mucho más amplia. Esta visualización forma parte de un estudio encabezado por la Universidad de Leuven, Bélgica. (Acar, 2014).

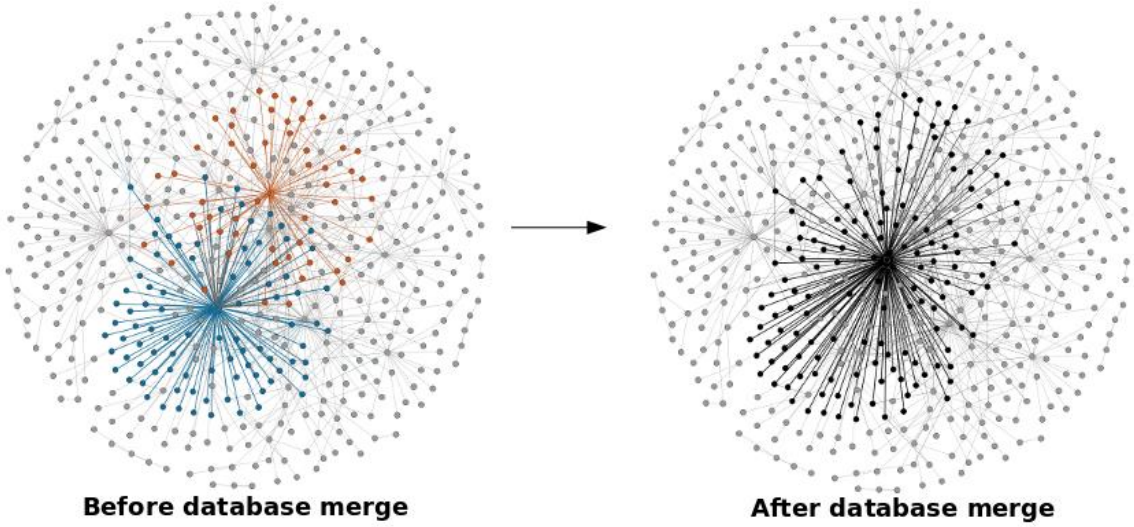


Figura 14 Visualización de la unión de bases de datos mediante cookie syncing

5.7 Detección de fingerprinting

Los scripts de análisis son cruciales en el desarrollo de este proyecto y el motivo que lo hace tan escalable e iterable a largo plazo analizando, el motivo es que una vez realizado el Crawling, todos los datos se encuentran en la base de datos y como consiguiente, pueden ser manipulados independientemente del crawl. Se decide realizar scripts de análisis de tal modo que las consultas a la base de datos sean automáticas y se puedan obtener los resultados deseados sin necesidad de analizar las bases de datos manualmente, de tal modo se puede automatizar y dejar la herramienta analizando la base de datos en búsqueda de los resultados deseados. Para esta primera parte del proyecto se han realizado cuatro scripts de análisis que como el resto del código se adjuntan cómo anexo.

Para detectar las técnicas de fingerprinting no es tan fácil como ver si se están usando las APIs usadas para tal fin o los métodos en concreto de tales APIs ya que estas APIs son usadas con muchos objetivos que nada tienen que ver con el fingerprinting. El método que tenemos para detectar el fingerprinting es la secuencia en que toman parte tales invocaciones de tal modo que no dan lugar a duda de la intencionalidad de generar un fingerprint. Cada tipo de fingerprint tiene su secuencia temporal de llamadas a las

APIs en función de la acción a realizar. En los siguientes apartados se van a detallar estas secuencias según el tipo de fingerprint.

Estos scripts pueden ser ejecutados independientemente analizando el contenido de una base de datos:

```
python script_analysis.py <dbname> <results_dir> <configfolder>
```

Añadir que, para todos ellos, se ha creado un cursor para recorrer la base de datos y buscar las secuencias de fingerprint con el objetivo de encontrar las secuencias que indican que hay fingerprint.
Finalmente, una vez se ha encontrado el fingerprint, se hace uso de la función creada

fingerprint2mongo del fichero insert2mongo también bajo el directorio de análisis.

canvas_analysis.py:

Este script realiza la función de detección de canvas fingerprinting. Como hemos vistos anteriormente en este documento, el canvas fingerprinting consiste en renderizar una imagen usando la API canvas. Obviamente su detección no es tan simple como observar la invocación de los métodos ya que esta API puede tener infinidad de usos que no sean hacer fingerprint al usuario, como por ejemplo una pizarra en que el usuario pueda dibujar entre otros muchos usos. Por lo que se analiza una secuencia de métodos de la API, que invocados en un determinado orden muestran una clara intención de rastreo.

El primer paso es realizar la query a la vista de la tabla JavaScript proveniente del crawl de tal modo que nos devuelva el conjunto de métodos de la API que son imprescindibles para generar la imagen del canvas, en este caso son 4:

```
CanvasRenderingContext2D.fillText,
CanvasRenderingContext2D.strokeText,
CanvasRenderingContext2D.fillStyle,
CanvasRenderingContext2D.strokeStyle
```

Todas las detecciones se añaden a una lista que es iterada para encontrar los posibles casos positivos de la secuencia necesaria. El primer paso es comprobar si la llamada es la primera llamada del script actual a la API, si se observa por otro lado que la anterior llamada era al mismo script, se parte de las detecciones anteriores para proseguir la secuencia. En caso de encontrar alguna de las llamadas con valores vacíos se descarta y se pasa a la siguiente llamada.

Posteriormente se deciden unos mínimos para concretar si se trata de fingerprinting, concretamente se decide un número mínimo de caracteres y el tamaño mínimo que por defecto hemos decidido marcar como mayor de 16x16 que sería el tamaño de un favicon y que se ha observado que canvas muchas veces es utilizado para crear esos favicons.

```
def large_enough(x_start, y_start, width, height):
 return width >= 16 and height >= 16
```

También se tienen en cuenta los números de estilos elegidos para ser representados en el canvas, si son inferiores a 2, la unicidad del fingerprint se reducirá mucho por lo que se puede descartar como una técnica de fingerprinting, el número de caracteres utilizados también es clave, para un valor inferior a 13 consideramos que su entropía es baja y no suficiente para identificar inequívocamente un usuario.

```
MIN_TEXT_STYLE_COUNT = 2 # Styles in use while writing text
MIN_CHARACTER_SET_SIZE = 13 # Number of distinct characters used
```

Es necesario también una vez generada la imagen, la obtención de sus valores, mediante la llamada a CanvasRenderingContext2D.getImageData dónde se devuelve el valor de los pixeles del canvas con lo que es indispensable para realizar el fingerprint.

Posteriormente, con todas las diferentes tuplas en sets, se buscan las interacciones o uniones en función de si son alternativas (una llamada sustituye a la otra) o si necesitan una de la otra como vemos a continuación.

```
candidates = urls_toDataURL.union(urls_getImageData)
candidates = candidates.intersection(urls_textWritten)
candidates = candidates.difference(urls_width.union(urls_height))
```

Finalmente, la lista de candidatos contendrá las tuplas (URL, script) resultantes que realizan fingerprint. Estos resultados mediante la función creada fingerprint2mongo de insert2mongo.py que usando la librería pymogno podemos subir valores a la base de datos MongoDB.

canvas_font_pg.py:
Para analizar canvas font fingerprinting, se tienen en cuenta dos nuevos métodos:

```
CanvasRenderingContext2D.font
CanvasRenderingContext2D.measureText
```

La propiedad CanvasRenderingContext2D.font de la API de Canvas especifica el estilo de texto que se utiliza al dibujar texto. Con la que de manera sencilla se puede obtener la lista de fuentes instaladas.

El método CanvasRenderingContext2D.measureText() devuelve un objeto que contiene información sobre el texto medido. Esta medida nos permite saber dos cosas, si uan fuente está instalada, ya que al no estar instalada tendrá la misma anchura que la fuente por defecto al dibujarse con esta, y la segunda información única del dispositivo como se detalla en el estado del arte.

Se ha considerado, viendo estudios sobre el uso de estas técnicas, que cualquier script que contenga estas dos llamadas y mida la longitud de las fuentes instaladas se puede asegurar que se está realizando canvas font fingerprinting.

Posteriormente, como en el resto de casos estos fingerprints son añadidos a la base de datos importando insert2mongo.py y su función fingerprint2mongo.

detect_webrtc.py:

Para detectar el web fingerprinting mediante webRTC, se analizan las llamadas a esta API de JavaScript en búsqueda del patrón de fingerprint. Para ello se analizan en concreto tres llamadas a la API webRTC, en concreto de la API RTCPeerConnection que se listan a continuación:

```
RTCPeerConnection.onicecandidate.
RTCPeerConnection.createDataChannel
RTCPeerConnection.createOffer
```

Con los resultados obtenidos del analysis de cada método, se crean varios Sets de Python con la URL y el script ejecutado.

```
cur.execute(("SELECT DISTINCT top_url, script_url FROM javascript_view "
 "WHERE symbol = 'RTCPeerConnection.onicecandidate';"))

onicecandidate = set()
for top_url, script_url in cur.fetchall():
 onicecandidate.add((top_url, script_url.split(' ')[0]))
```

Posteriormente se busca la intersección de los diferentes sets creados, obteniendo entonces los resultados como las tuplas URL, Script que han ejecutado todo el proceso.

detect_audio.py:
El script de detección de audio realizado es muy similar al de webRTC ya que se busca los procesos dónde se han ejecutado secuencias completas. En este caso la configuración
de audio fingerprinting es la siguiente:

- AudioContext.createOscillator
- AudioContext.createDynamicsCompressor
- AudioContext.destination
- AudioContext.createAnalyser
- AudioContext.createGain

Y sus respectivos homólogos en OfflineAudioContext que a pesar de renderizar diferentemente, ambos pueden ser usados como fingerprint. Como vemos hay que analizar dos métodos diferentes para generar el fingerprint.

Como en el caso anterior, se crean sets donde se almacenan las respuestas de las consultas a la base de datos del crawl para a posteriori, crear las intersecciones de los diferentes Sets, de tal manera que podamos ver los casos en que se produce toda la cadena de llamadas a la API necesarias para realizar fingerprints.

5.8 Análisis de fingerprinting de segunda generación en la web: WebGL

Otra API de HTML5 muy usada para rastrear mediante fingerprinting, que es utilizada para renderizar graficos 2D y 3D, es WebGL. Su uso es muy similar al del canvas, ya que se renderiza una imagen con pequeñas variaciones debido a la GPU de cada dispositivo.

A diferencia del canvas fingerprinting esta técnica nace a posteriori, debido a dificultades técnicas que la hacían inviable para cualquier tipo de fingerprinting dado que existe a priori un grado de incertidumbre debido a parámetros como el aliasing.

Varios equipos, demostraron que controlando estos parámetros se podía conseguir realizar imágenes únicas y estables para cada dispositivo.

Los equipos de MaxMind y Augur realizaron las primeras soluciones comerciales con tal fin. Su método consistía en dibujar un gradiente con sombrasy texturas y convertir esa imagen resultante a base64 para generar un hash enorme único para cada renderización para cada gráfica en cada configuración. Para generar más entropía a este se le une información de la propia API asociada a cada versión de ella misma. Cómo son las extensiones de WebGL y sus posibilidades activas.

Este primer sistema de fingerprinting hace uso de las llamadas a los métodos siguientes para forzar a la GPU a renderizar texturas complicadas de manera única.

- WebGLRenderingContext.createBuffer
- WebGLRenderingContext.bindBuffer
- WebGLRenderingContext.bufferData
- WebGLRenderingContext.createProgram
- WebGLRenderingContext.createShader
- WebGLRenderingContext.shaderSource
- WebGLRenderingContext.compileShader
- WebGLRenderingContext.attachShader
- WebGLRenderingContext.linkProgram
- WebGLRenderingContext.useProgram
- WebGLRenderingContext.getAttribLocation
- WebGLRenderingContext.getUniformLocation
- WebGLRenderingContext.enableVertexAttribArray
- WebGLRenderingContext.uniform2f
- WebGLRenderingContext.drawArrays

Se ha realizado el script de análisis para encontrar esta cadena de llamadas a la API de WebGL y encontrar que sitios web están utilizando este método.

Posteriormente, Yinzhi Cao, professor de la universidad de Lehigh, y su equipo llevaron el proceso un paso más hacia delante y migraron esta solución que por el nivel de entropía que ofrece ya de por si es una solución muy potente a una solución de rastreo

no de navegadores, sino a nivel de hardware independientemente de quien lo ejecute, lo que es denominado como Cross-browser fingerprinting.

Esto es posible dado que, en vez de capturar la imagen resultante de la renderización, utilizan la generación de las texturas en los buffers previo procesado de los navegadores, que tienen sus métodos de suavizado de bordes, de filtrado antialiasing y otras propiedades para mejorar la visualización que rompen esta estabilidad entre navegadores.
La principal propiedad única de las tarjetas gráficas que es usada es el algoritmo de interpolación que difiere en distintos tipos de hardware y permite identificar al dispositivo. Para ello las texturas se generan con degradados ya que es el único modo de explotar esa propiedad.

Las GPU usan el "alpha channel", para nativamente mediante drivers propios del hardware, decidir el nivel de transparencia en función de una Alpha entre 0 y1. Que esta propiedad está ligada a la GPU y no al navegador, es lo que nos permite identificar el hardware y no evaluar el conjunto junto al navegador.

Tras leer el paper sobre el desarrollo que habían realizado, y tras observar la factibilidad la posibilidad de que la industria estuviera haciendo uso de esta técnica para rastrear mejor a los usuarios, decidí realizar un script de análisis de este tipo de fingerprint.

Tuve la posibilidad de hablar con el equipo de la universidad de Lehigh, y colaboramos para encontrar una secuencia de métodos de diferentes APIs HTML5 que identificaran este tipo de fingerprinting y lo distinguiesen del que no era cross-browser para de tal modo poder observar si la industria estaba sacando partido de esa investigación realizada en Lehigh.

Del propio script de análisis podemos extraer las siguientes llamadas a la API de WebGL.

- WebGLRenderingContext.createShader(); // creamos un shader para vertices o frames
- WebGLRenderingContext.bindTexture(); //confirmamos que se están usando texturas
- WebGLRenderingContext.readPixels(); // el script debe leer la imagen resultante en el canvas para generar el fingerprint
- Canvas.getDataURL(); // el script debe leer la imagen resultante en el canvas para generar el fingerprint

Posteriormente en el apartado de resultados evaluamos los usos de la herramienta.

5.9 Análisis de las herramientas de investigación obtenidas, sus usos y resultados.

La herramienta de crawling diseñada nos permite obtener resultados que, de otra manera, mediante estudios automatizados no sería posible de obtener debido al análisis estático realizado por las mismas y que por análisis no automático no sería viable de obtener debido al elevado tiempo de análisis que requeriría.

Con la herramienta actual, se realiza el análisis de 400.000 sitios web para hallar las técnicas más utilizadas e identificar tanto los sitios que utilizan fingerprinting como los scripts y empresas utilizadas para tal fin al mismo tiempo que se obtienen datos demográficos, se sacan insights de los mismos.

Otro de los objetivos de la herramienta era realizar un estudio mensual de las técnicas halladas para evaluar como progresa en el tiempo el uso de las diferentes técnicas estudiadas.

Estos van a ser estudiados en el último apartado de resultados y conclusiones en detalle para incluir también el análisis de los resultados de la segunda parte el proyecto basados en el uso de la herramienta diseñada para el caso particular de minado de criptodivisas en el navegador.

5.10 "Stateless" Script de identificación de usuarios mediante técnicas de fingerprinting

Por motivos de propios de análisis de la herramienta y usos internos de investigación dentro de Telefónica tenía sentido realizar un script basado en estas técnicas para generar nuestro propio fingerprint.

Para ello no he partido de cero, se ha estudiado y analizado la librería realizada por Valentin Vasilyev (@Valve) un ingeniero ucraniano que con la ayuda de la comunidad de Github lleva 5 años trabajando para la generación de una librería de fingerprinting fiable y con alta entropía para poder identificar inequívocamente a todos los dispositivos y sus navegadores.

El objetivo que perseguimos al realizar este script es identificar mejor las diferentes formas de fingerprinting al estudiar la manera en que son generadas y no únicamente la detección. Filtrar las propiedades que pueden ser variantes como las propiedades de un monitor, el hecho de tener extensiones instaladas como adblock (esta información pasa a añadirse como complementaria en nuestro sistema, pero no forma parte de la firma). Añadir nuevas técnicas de fingerprinting que no están disponibles en la librería. Posibilitar el uso de técnicas a síncronas, cosa que la librería no permite por lo que se ha rediseñado el core para permitir técnicas asíncronas como son la mayoría de técnicas de fingerprinting y en este caso, todas las implementadas.

Con técnica genera una key que posteriormente, se utiliza un generador de hashs de 126 bits dado que nuestro script es capaz de darnos una entropía de alrededor de 32 bits aseguramos que no reducimos la misma al generar los los hashs. Los hashes de las keys se van calculando sobre la anterior key para de tal modo obtener el hash final.

```
results.push(this.pluginsKey())
results.push(this.canvasKey())
results.push(this.WebRTCKey()) //needs to be a promise
results.push(this.webglKey())
results.push(this.webglVendorAndRendererKey())
results.push(this.adBlockKey())
results.push(this.hasLiedLanguagesKey())
results.push(this.hasLiedResolutionKey())
results.push(this.hasLiedOsKey())
results.push(this.hasLiedBrowserKey())
results.push(this.touchSupportKey())
results.push(this.customEntropyFunction())
results.push(this.audioKey()) //needs to be a promise
results.push(this.fontsKey()) //needs to be a promise
```

Figura 15 Tozo de código dónde se pueden apreciar técnicas que hacen indispiensable el uso de llamadas asincronas (resuletas mediante js promises)

A continuación, se comentan algunas de las técnicas añadidas más interesantes dada su estado actual de estudio más limitado respecto al resto.

WebRTC fingerprinting:

Este es un caso muy sencillo que nos permite entender la necesidad de código asíncrono, dado que se usa webRTC para falsear una conexión remota y obtener la IP privada del usuario. Al establecer esta conexión quedamos a la espera de que se establezca por lo que necesitamos algún tipo de alerta que nos notifique cuando ha se ha obtenido el valor deseado de vuelta por lo que hacemos uso de las Promises de JavaScript que nos permiten usar resolve para obtener el resultado de la técnica comentada. Aunque a priori parezca que no es determinante conocer la IP privada del usuario, puede ser muy interesante conocer a en entornos empresariales para distinguir dispositivos muy parecidos. Dada la volatilidad de la misma, esta técnica no se ha añadido al fingerprint final, sino que se deja como parte de la telemetría adjunta que nos permite conocer más información del usuario.

```
getWebRTCFp: new Promise((resolve, reject) => {
  window.RTCPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection || window.webkitRTCPeerConnection;
  var pc = new RTCPeerConnection({iceServers:[]}), noop = function(){};
  var privateIP;
  pc.createDataChannel("");    //create a bogus data channel
  pc.createOffer(pc.setLocalDescription.bind(pc), noop);    // create offer and set local description
  pc.onicecandidate = function(ice){  //listen for candidate events
      if(!ice || !ice.candidate || !ice.candidate.candidate)  reject();
      privateIP = /([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-f0-9]{1,4}(:[a-f0-9]{1,4}){7})/.exec(ice.candidate.candidate)[1];
      pc.onicecandidate = noop;
      resolve({key: 'webrtc', value: privateIP, doNotAdd: true});
  };
}),
```

De esta implementación asíncrona como efecto colateral hemos conseguido reducir el tiempo total de ejecución del script para obtener el resultado del fingerprint a pesar de añadir nuevas técnicas.

WebGL fingerprinting:

La huella dactilar de WebGL es una combinación de técnicas, que se encuentra en el script antifraude de MaxMind y en en el fingerprint de Augur.io.

Primero dibuja un objeto degradado con sombreadores y conversa la imagen con la cadena Base64.

Luego enumera todas las extensiones y capacidades WebGL y las agrega a la cadena Base64, lo que da como resultado una enorme cadena WebGL, potencialmente muy exclusiva y singular de cada dispositivo.

Es interesante saber que el canvas utilizado en ningún momento es renderizado de tal maenra que pasa completamente inadvertido para el usuario.

```
gl.drawArrays(gl.TRIANGLE_STRIP, 0, vertexPosBuffer.numItems)
if (gl.canvas != null) { result.push(gl.canvas.toDataURL()) }
result.push('extensions:' + gl.getSupportedExtensions().join(';'))
result.push('webgl aliased line width range:' + fa2s(gl.getParameter(gl.ALIASED_LINE_WIDTH_RANGE)))
result.push('webgl aliased point size range:' + fa2s(gl.getParameter(gl.ALIASED_POINT_SIZE_RANGE)))
result.push('webgl alpha bits:' + gl.getParameter(gl.ALPHA_BITS))
result.push('webgl antialiasing:' + (gl.getContextAttributes().antialias ? 'yes' : 'no'))
result.push('webgl blue bits:' + gl.getParameter(gl.BLUE_BITS))
result.push('webgl depth bits:' + gl.getParameter(gl.DEPTH_BITS))
result.push('webgl green bits:' + gl.getParameter(gl.GREEN_BITS))
result.push('webgl max anisotropy:' + maxAnisotropy(gl))
result.push('webgl max combined texture image units:' + gl.getParameter(gl.MAX_COMBINED_TEXTURE_IMAGE_UNITS))
```

Figura 16 Parte del código para la generación el canvas mediante WebGL

Audio Fingerprinting:

Para realizar este tipo de técnica se genera una señal mediante la API de AudioContext que nos permite crear un oscilador, en este caso usamos una señal triangular dado que en los cambios bruscos es dónde más irregularidades propias de cada dispositivo se pueden apreciar. Finalmente conectamos la salida del mismo al un buffer interno de tal modo que no vamos a sacar el audio por la salida de audio del dispositivo de tal manera que es completamente imperceptible al usuario del mismo modo que el resto de técnicas estudiadas.

```
if (context = new(window.OfflineAudioContext || window.webkitOfflineAudioContext)(1, 44100, 44100), !context) {
    set_result("no_fp", "pxi_result");
    pxi_output = 0;
}

// Create oscillator
var pxi_oscillator = context.createOscillator();
pxi_oscillator.type = "triangle";
pxi_oscillator.frequency.value = 1e4;

// Create and configure compressor
var pxi_compressor = context.createDynamicsCompressor();
pxi_compressor.threshold && (pxi_compressor.threshold.value = -50);
```

Figura 17 Parte del código para la generación de la señal de audio usando la API de AudioContext

El total de las diferentes propiedades del navegador y técnicas usados son las siguientes:

1. UserAgent
2. Language
3. Color Depth
4. Screen Resolution
5. Timezone
6. Has session storage or not
7. Has local storage or not
8. Has indexed DB
9. Has IE specific 'AddBehavior'
10. Has open DB
11. CPU class
12. Platform
13. DoNotTrack or not
14. Full list of installed fonts (maintaining their order, which increases the entropy), implemented with Flash.
15. A list of installed fonts, detected with JS/CSS (side-channel technique) - can detect up to 500 installed fonts without flash
16. Canvas fingerprinting
17. WebGL fingerprinting
18. Plugins (IE included)
19. Is AdBlock installed or not
20. Has the user tampered with its languages
21. Has the user tampered with its screen resolution
22. Has the user tampered with its OS
23. Has the user tampered with its browser
24. Touch screen detection and capabilities
25. Pixel Ratio

26. System's total number of logical processors available to the user agent.
27. webRTC
28. Audio fingerprinting

Para utilizar la librería creada tenemos que llamar al script.

```
<script src="Fingerprint.js"></script>
```

Instanciar la librería y utilizar el método get para obtener el fingerprint.

```
    new Fingerprint().get(function(result){
        console.log(result);
    });
```

El resultado que obtenemos tendrá el siguiente formato:

```
{
    "fp":"81b157c3c978490c97faa18ac1f01f8b",
    "timestamp":1510918102414,
    "resolution":[1280,800],
    "resolution_available":[1280,724],
    "user_agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36",
    "osname":"macOS",
    "osversion":"10.12.6",
    "browser":"Chrome",
    "browserversion":"62.0"
}
```

Como vemos el campo fp, es el identificador único que usamos para identificar al usuario. El resto, son parámetros propios del navegador que pueden variar con mayor facilidad. Como la versión del sistema operativo, la resolución de la pantalla o la versión del navegador entre otros.

De tal manera que al diferenciar ambos conseguimos una entropía inferior en la firma pero mucho más estáticos en el tiempo lo que nos da más exactitud en los resultados. La importancia de estos campos para telemetría hace que se consideren importantes y se adjunten como complementarios. Estos campos variables se pueden estudiar para sacar conclusiones.

A continuación, procederemos al estudio de algunas de las técnicas implementas y la metodología para generar y anexionar los diferentes fingerprints y parámetros para generar el fingerprint final.

Una de las metodologías de fingerprinting que se quería añadir y que finalmente ha sido desestimada ha sido una implementación del desarrollo del WebGL fingerprinting cuya diferencia con el que he desarrollado, es que debe permitir la identificación independientemente del navegador que se use en el dispositivo, lo que es conocido como cross-browsing fingerprinting, después de muchos tests y de hablar con Yinzhi Cao, profesor de la universidad de Lehigh, hemos llegado a la conclusión que esta técnica de fingerprinting, es muy inestable y no nos va a permitir obtener los resultados con la estabilidad y fiabilidad deseada.

Para poder estudiar los resultados, como se comenta anteriormente se crea una base de datos que aparte de poder almacenar información complementaria como la dirección IP pública del usuario que nos permite obtener más datos de telemetría del usuario, nos permite llevar un recuento de las veces que se ha producido un positivo y con que fingerprint ha sido la nueva muestra.

Por ejemplo, para un dispositivo concreto con un navegador concreto utilizado para las pruebas tenemos muchos positivos con un mismo fingerprint y analizando el resto de valores obtenidos vemos cómo no ha habido ningún falso positivo en ese sentido.

En la siguiente figura podemos ver resultados obtenidos con un mismo fingerprint desde un mismo dispositivo y mismo navegador, cada uuid identifica un positivo distinto con un mismo identificador.

Analizando la primera muestra obtenida y la última y comparando sus timestamps vemos que han transcurrido un total de 13 días por lo que también podemos asegurar que es un método de rastreo sin estado a diferencia de las cookies muy válido y muy difícil de detectar.

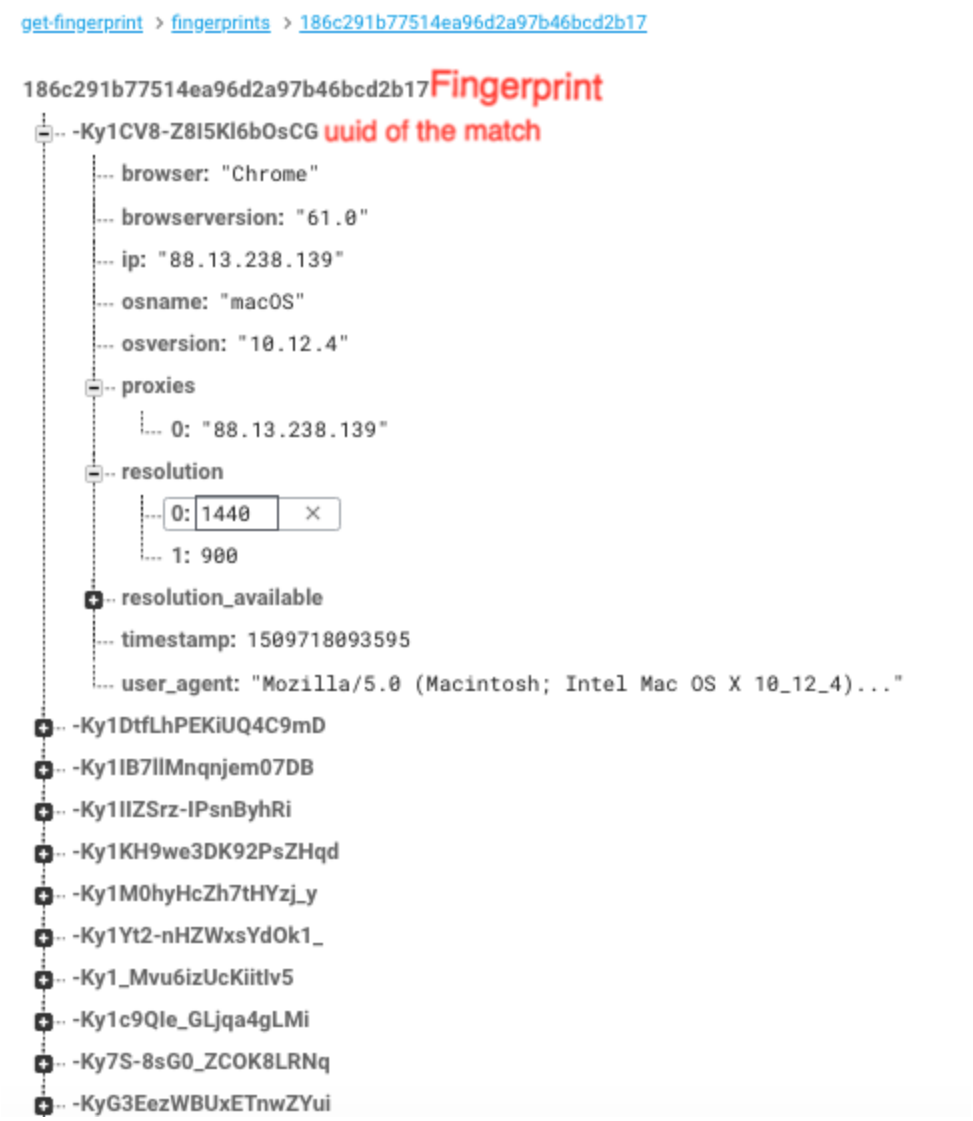


Figura 18 Estructura de lso fingerprints almacenados en la base de datos

En la siguiente figura vemos como dispositivos que a priori son muy parecidos en hardware y software (Android), pueden ser distinguidos inequívocamente por nuestra librería de fingerprinting mejorando de manera muy notable en este caso los resultados obtenidos con la versión opensource del script.

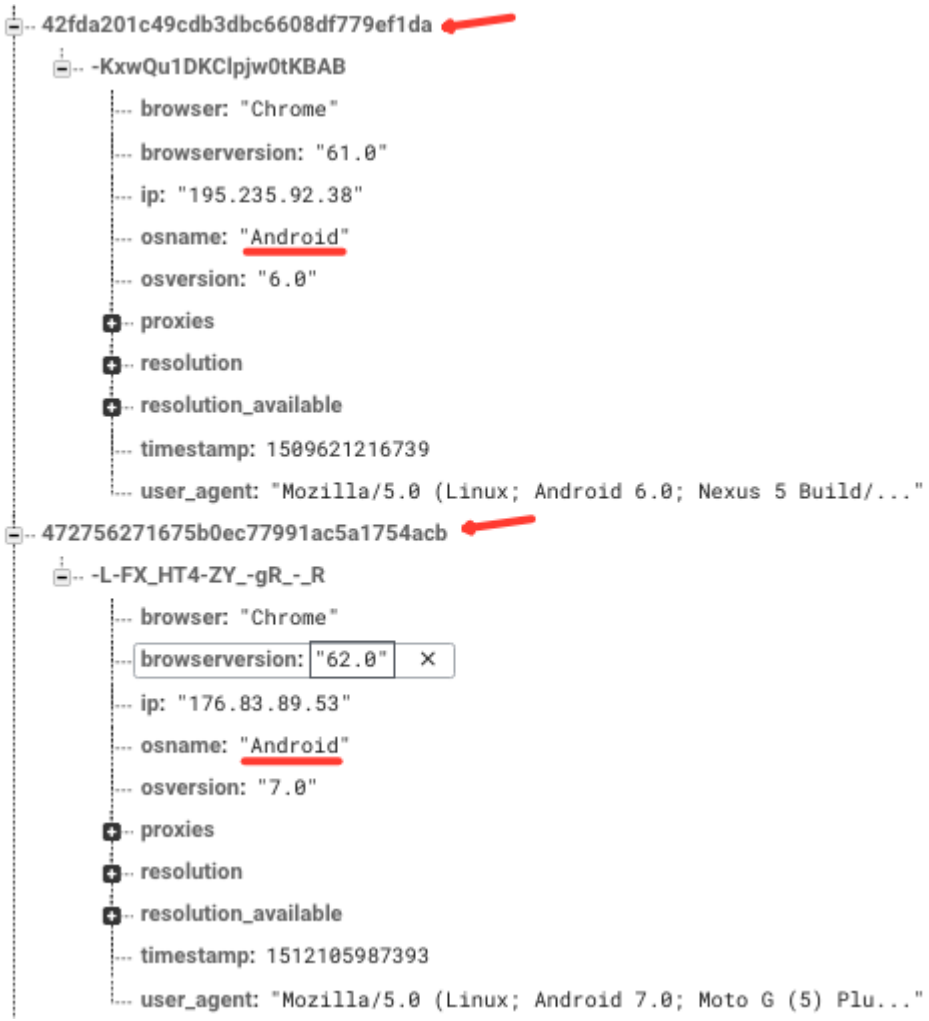


Figura 19 Ejemplo dispositivos parecidos e identificados inequívocamente

Este experimento nos ha llevado a construir una solución muy potente que puede ser usada para muchos casos de uso distintos como obtener telemetría de un usuario para asegurar de manera más certera que se trata de él mismo.

Esta herramienta también ha sido usada para testear nuestras soluciones para detección de las mismas con la herramienta de crawling y analizar posibles modificaciones que pueden llegar a ser realizadas para no ser detectadas por nuestro patrón de detección. Cosa que de otra manera debido a la ofuscación utilizada en la industria conllevaría tiempos de análisis muy superiores.

6. Análisis de minado de criptomonedas en navegadores

6.1 Estado del arte del minado de criptomonedas

Los problemas a los que los usuarios pueden enfrontarse mientras navegan por internet no hacen referencia única y exclusivamente a al uso de sus datos personales para ser identificados y como consecuencia ser su privacidad invadida.

El minado de criptomonedas es una industria que en los últimos años ha demostrado ser muy lucrativa. A grandes rasgos el minado de criptomonedas es el proceso de validación de transacciones hechas y añadirlas a la cadena de bloques (blockchain) de esa particular moneda después de ser comprobadas por varias técnicas, en general PoW (proof of work) o PoS (proof of steak), este proceso para por ejemplo para el caso del bitcoin, se trataría de un Proof-of-Work cuyo objetivo es encontrar un Nonce para el SHA-256 del bloque sea menor que el del bloque marcado por la dificultad y en caso de ser el primero en hallarlo serás el recibidor de un una recompensa por dicha validación.

Este proceso es costos a nivel de recursos necesarios para validar estas transacciones, que a posteriori son remuneradas con la creación de nuevas monedas que, cómo acabamos de comentar son pagadas a las personas que han validado esas transacciones (y aunque no vamos a entrar en detalles, el resto de usuarios tratando de validar esa transacción tienen ese coste computacional Asociado al Proof-of-Work en este caso de Bitcoin pero no reciben la recompensa).

El algoritmo de resolución, puede depender de muchos factores y en concreto el tipo de función de hashing puede variar mucho el tipo de dispositivo adecuado usado para minar las criptomonedas. Por ejemplo en el caso de Bitcoin, como se ha comentado anteriormente, el Poof-of-Work lo resuelve el que más potencia de computación tenga ya que la función SHA-256 se resuelve mediante operaciones matemáticas, que al poderse resolver de manera concurrente en paralelo cosa para que las GPUs son muy buenas ya que están acostumbradas a realizar esas operaciones para grupos de pixeles y similares por lo que su diseño está pensado para realizar tareas simples de manera muy rápida y concurrente (disponen de más Unidades Lógicas Algorítmicas (ALUs)) que la CPU. En concreto existen placas ASIC hechas específicamente para este fin que son mucho más potentes que las GPUs.

Para hacernos una idea, con la llegada de las ASICs especializadas en minar Bitcoins, el poder de hashing de la red aumento y con ello la dificultad hasta el punto que para validar un único bloque un ordenador normal usando la CPU minando a 10MH/s tardaría 425 años.

En la siguiente imagen podemos ver como el crecimiento del interés por el Bitcoin (BTC) y como consecuencia su precio de mercado ha hecho crecer de manera proporcional el precio de las acciones de Nvidia, la empresa cuyas GPUs **comerciales** son más usadas y tenen un rendimiento económico superior.

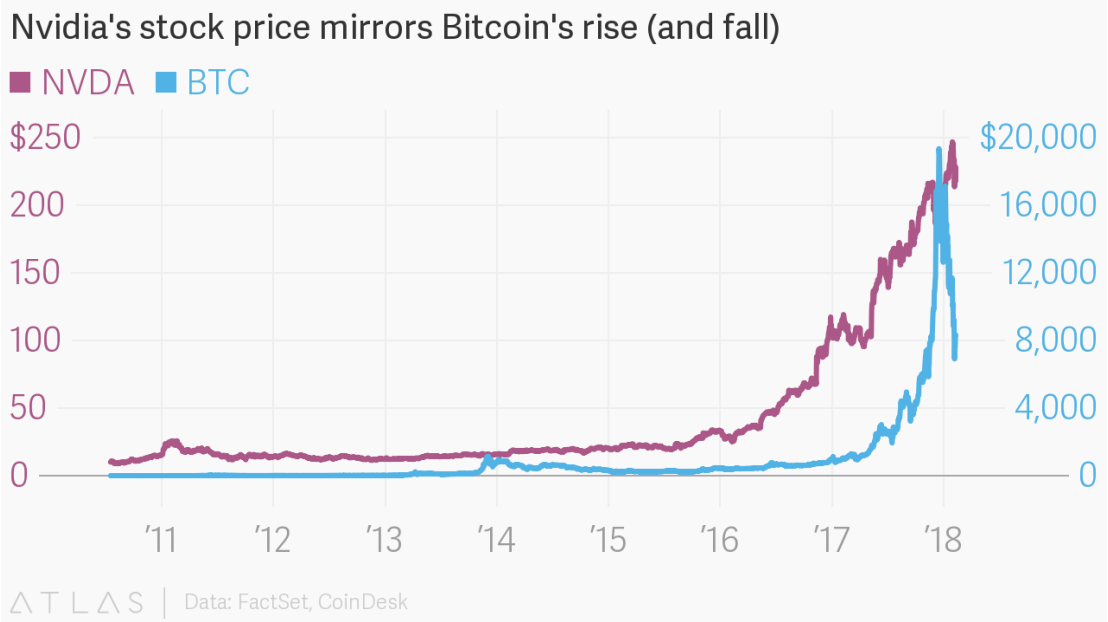


Figura 20 Precio de las acciones de Nvidia vs Precio del Bitcoin

De este gráfico podemos extraer y concuir que el miando una operación rentable a nivel de costes siempre que dispongamos de una máquina potente que nos permita ser el primero en resolver bloques lo que no ocurriria con hardware menos potente y por lo cual su uso quedaba restinjido a grandes máquinas pensadas para tal fin.

Para entender por qué ha proliferado el uso de mineros en navegadores, es necesario tener en cuenta los pilares lo hacen un segmento interesante, y en general son los que vienen marcados por las propiedades de una moneda en particular el Monero(XMR).

Figura 21 Datos de la valoración de la criptomoneda Monero y su capitalización

El caso de Monero es distinto ya que utiliza un algoritmo de hashing distinto que está diseñado para conseguir un PoW igualitario, esta función de hashing igualitario es

conocida como CryptoNight y ha sido diseñada por el mismo equipo de CryptoNote. Esta función de hashing tiene como diferencia que es memory-hard, lo que significa que a diferencia de SHA-256, utiliza mucha memoria en el proceso de generación del hash.

CryptoNight como algoritmo igualitario de resolución de hashes, a diferencia de SHA-256 no permite que los dispositivos más rápidos en resolver dichas operaciones sean los que siempre van a resolver el PoW. Eso es debido a que la dificultad del algoritmo CriptoNight reside en la cantidad de memoria necesaria y no en la complejidad de sus operaciones. Almacena muchas operaciones en memoria y necesita acceder a ellas por lo que este es el cuello de botella del algoritmo.

Muy pocos dispositivos disponen de la memoria necesaria para utilizar el algoritmo CryptoNight (alrededor de 2MB) a la velocidad necesaria para ser eficientes en la realización de las tareas necesarias para obtener el hash resultante. Este algoritmo está pensado para que sea poco eficiente en GPUs o ASICs por lo no se consiguen grandes diferencias dependiendo del ordenador o maquina usada para minar con esta técnica lo que lo hace un algoritmo igualitario.

```
// Parts of this file are originally copyright (c) 2012-2013 The Cryptonote developers

#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

#include "common/int-util.h"
#include "hash-ops.h"
#include "oaes_lib.h"

#define MEMORY         (1 << 21) // 2MB scratchpad
#define ITER           (1 << 20)
#define AES_BLOCK_SIZE  16
#define AES_KEY_SIZE    32
#define INIT_SIZE_BLK   8
#define INIT_SIZE_BYTE (INIT_SIZE_BLK * AES_BLOCK_SIZE)
```

Figura 22 Cabecera del algoritmo CriptoNight dónde se puede apreciar que requiere 2MB de memoria reservada

```
940     /* CryptoNight Step 2:  Iteratively encrypt the results from Keccak to fill
941      * the 2MB large random access buffer.
942      */
943
944     aes_expand_key(state.hs.b, expandedKey);
945     for(i = 0; i < MEMORY / INIT_SIZE_BYTE; i++)
946     {
947         aes_pseudo_round(text, text, expandedKey, INIT_SIZE_BLK);
948         memcpy(&hp_state[i * INIT_SIZE_BYTE], text, INIT_SIZE_BYTE);
949     }
```

Figura 23 Podemos ver que en el segundo paso del algoritmo de hashing rellena la memoria con el resultado del encriptado inicial

```
973     /* CryptoNight Step 4:  Sequentially pass through the mixing buffer and use 10 rounds
974      * of AES encryption to mix the random data back into the 'text' buffer.  'text'
975      * was originally created with the output of Keccak1600. */
976
977     memcpy(text, state.init, INIT_SIZE_BYTE);
978
979     aes_expand_key(&state.hs.b[32], expandedKey);
980     for(i = 0; i < MEMORY / INIT_SIZE_BYTE; i++)
981     {
982         // add the xor to the pseudo round
983         aes_pseudo_round_xor(text, text, expandedKey, &hp_state[i * INIT_SIZE_BYTE], INIT_SIZE_BLK);
984     }
```

Figura 24 Podemos ver como se usa la memoria para mezclar mediante AES el contenid odel buffer

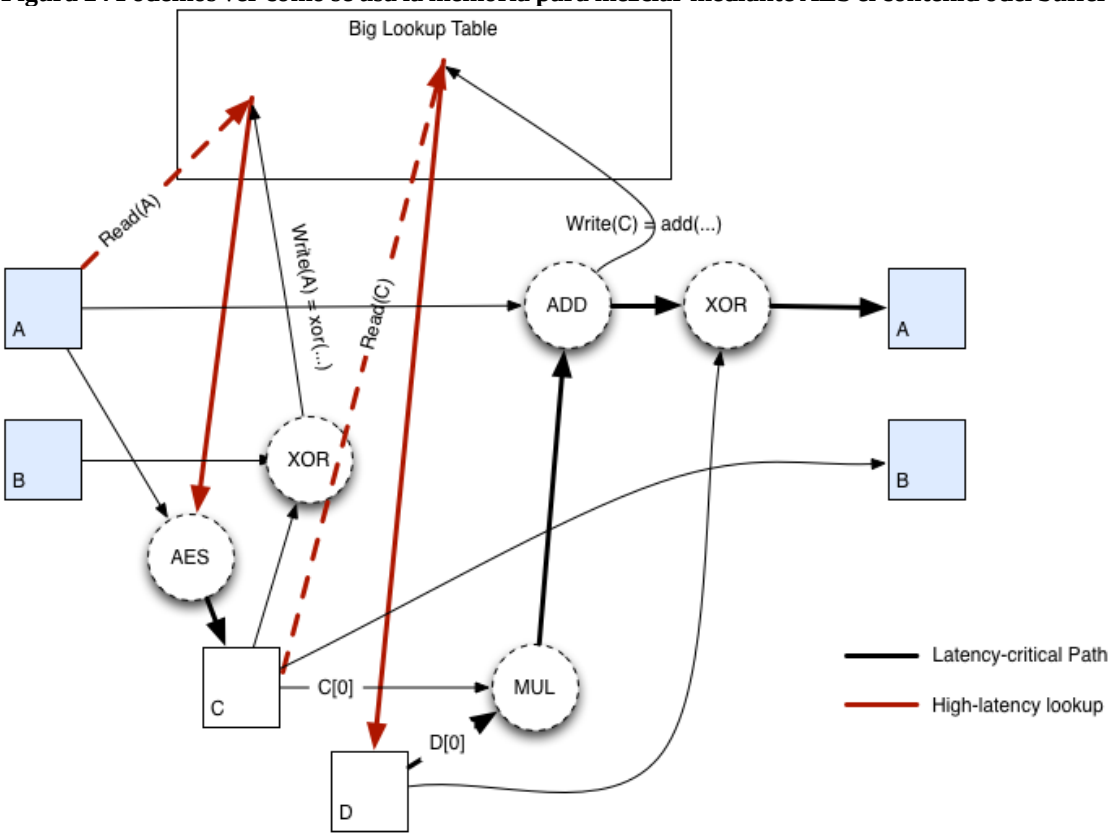


Figura 25 Diagrama a alto nivel del algoritmo CryptoNight que nos permite ver como las búsquedas en memoria son los pasos con más latencia

Este algoritmo es una función de hashing, por lo que su objetivo es obtener una cadena binaria de longitud fija a partir de una entrada variable tanto en tamaño como en tipo. Para que sea útil tiene que ser relativamente rápido resolviendo el hash y no previsible sin el uso del algoritmo en sí.

El algoritmo se basa como hemos visto en el código y el diagrama superior en un espacio de memoria de 2MB también denominado scratchpad, que se usa para almacenar los pasos intermedios de la operación como vemos en la figura 21, captura del código fuente del algoritmo.

El algoritmo consta principalmente de tres partes.

1. Preparar la tabla en memoria
2. Ejecución del bucle principal de cálculo (cuello de botella del proceso debido al uso de la memoria)
3. Cálculo del resultado final

Antes de pasar a la primera fase sirve para inicializar el estado de un bloque de datos de 200Bytes que va a ser la semilla del resultado de nuestro algoritmo(CryptoNight). Para ello se utiliza un algoritmo de hashing inicial llamado keccak como generador de ese bloque inicial con una serie de parámetros iniciales.

Se parte de los 32 bytes de mayor peso de la salida del algoritmo de keccak y se expanden a 10 keys de AES que son usadas para las posteriores rondes de ejecución de AES con los 8 bloques de 16 bytes del resultado del algoritmo de keccak. El resultado del hash mediante AES se va almacenando en memoria y se usa el valor obtenido como semilla para generación del siguiente bloque hasta rellenar los 2097152 bytes (2MB) de memoria reservados.

La segunda etapa es la que requiere de más lecturas y escrituras en memoria que es lo que finalmente lo hace un algoritmo igualitario y el que vemos explicado en el diagrama de la figura 22. Lo que este algoritmo realiza es la separación de los bits de mayor peso y menor peso en dos buffers A y B, estos segundos pasados por una XOR.

A posteriori con estas dos variables se desencadena el bucle que se ejecuta 524288 veces. Se usan las variables A, B, C y D como direcciones de memoria y se van leyendo y modificando y como hemos comentado ya anteriormente esta lectura va a ser el cuello de botella del algoritmo. El algoritmo que podemos observar se puede dividir en general en 6 fases:

1. Se convierte A a una dirección de memoria de nuestra tabla con la función que hemos definido Read(A) en el diagrama.
2. Se aplica AES al contenido en la dirección obtenida de A de memoria en la etapa anterior usando A como key y se guarda el resultado en una variable C.
3. C se guarda en B pero antes se utiliza el valor anterior de B para aplicarle un XOR con el valor de la etapa 2 y guardarlo en la posición de memoria de la etapa 2. Ambas operaciones deben hacerse a la vez porque cada una pisa la otra y es el motivo del uso de las variables C y D lo que aún aumenta más el uso de memoria y las lecturas a la misma.

4. Se convierte B a una dirección de memoria que vamos a llamar D.
5. Se obtiene una nueva a haciendo la suma de A y el resultado de la multiplicación de B y D.
6. Se obtiene una nueva a desde el XOR de A con D.

   Este algoritmo es cíclico y va a ser ejecutado 5244288veces, y las salidas de A y B son las que van a ser usadas en la siguiente iteración.

Finalmente, con todos los valores en memoria, se procede a la obtención del resultado. Que consisten en usar la función de hashing keccak para obtener una cadena de bits que de la que se va a obtener sus dos bits de menor peso y en función del valor que tengan se va a proceder a usar una función de hashing final.

Valor	Función
00	Blake-256
01	Groestl-256
10	Jh-256
11	Skein-256

Tabla 3 Función de hashing resultante elegida en función de la salida de la función de keccak

Finalmente, a la función elegida se le pasa la salida completa que acabamos de obtener de la función de keccak y ese es el hash resultante del Algoritmo de CriptoNight.

La conclusión que podemos sacar de este algoritmo es que al usar constantemente memoria como hemos ido comentando, la rapidez al realizar las funciones de hashing no es determinante por lo que las técnicas que hagan uso del mismo serán un claro candidato a ser ejecutados en ordenadores domésticos y sin acceder a la GPU.

Otra de las ventajas de Monero es su privacidad en cuanto a la trazabilidad de las transacciones usando el método firmas de anillo (ring signatures), por la que un grupo de mineros forman un anillo y uno de ellos el que ha generado la transacción, pero es indetectable cuál del grupo la ha generado por lo que en un supuesto caso que se quisiera saber qué empresa ha estado validando esas transacciones o quien ha generado los pagos es algorítmicamente inviable saber quién ha generado o validado una transacción.

Estas dos propiedades son las que hacen de Monero la moneda más interesante de minar en el navegador y como veremos la más utilizada por las empresas dedicadas a ello.

6.2 Estado del arte del minado en el navegador

El minado de criptomonedas en el navegador está en auge es un tema en auge desde la segunda mitad de 2017 y el sector de las empresas de la seguridad informática la ven como la principal amenaza de este 2018. Definido en la revista heise.de “la nueva mierda del crimenware” ofrece a quien lo utilize la posibilidad de obtener dinero por nada a cambio.

Principalmente existen dos tipos de mineros considerados como web miners, los ejecutados en el navegador y los que son ejecutados en el dispositivo mediante una descarga en el navegador. En este estudio vamos a centrarnos en los que propiamente se ejecutan sobre el navegador ya que la problemática de los instalados en el propio dispositivo hace referencia a problemas de seguridad propios del dispositivo y no una problemática relacionada con el minado de criptomonedas.

En este análisis nos vamos a centrar en el minado en el propio navegador porque a priori no está surgiendo ninguna falla de seguridad y es por ello que es más complicado que sean identificados, estos mineros que se ejecutan en el navegador son mineros de Moneros previamente analizados con detalle. Este tipo de minado se ejecuta en el navegador del usuario y usa los recursos del mismo para realizar las operaciones necesarias para el PoW y resolver bloques.

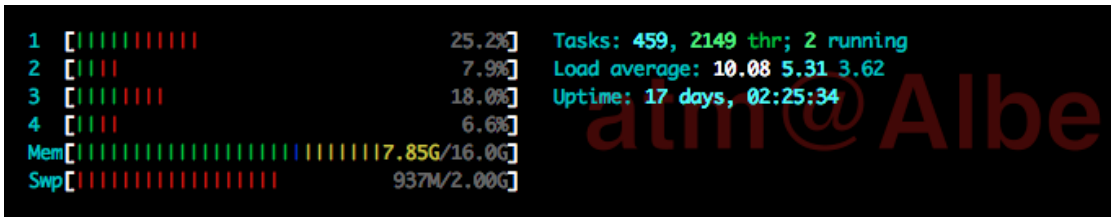


Figura 26 CPU sin minar

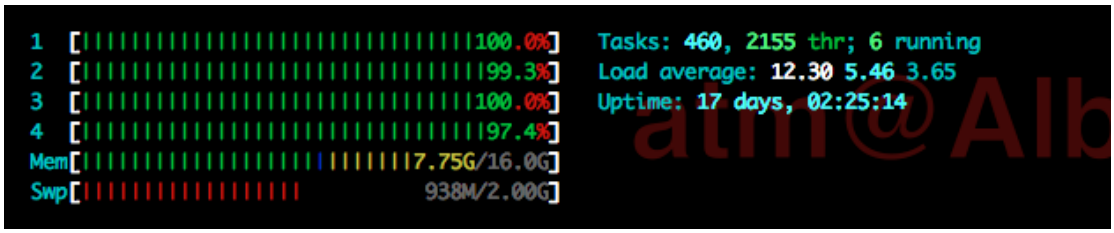


Figura 27 CPU minando

El minado en el navegador ha ido evolucionando en los últimos años desde la aparición del primer servicio profesional con tal fin en septiembre de 2017, llamado Coinhive y la consiguiente aparición de alternativas en el sector hasta el punto de existir decenas de empresas dedicadas ofreciendo distintos servicios de minería de Moneros.

Es interesante ver como el minado de Moneros en el navegador no es una operación rentable. Si nos fijamos en el experimento realizado en la revista C'T (c't 9/2018, p. 83), podemos ver que el caso concreto del servicio principal que se va a estudiar, el coste de minar durante una hora en un ordenador estándar repercute en un beneficio que si convertimos a euros equivaldría a 0.0016 euros, y el coste en electricidad de semejante rendimiento es de 0.03 euros. Por lo que podemos ver el coste del minado prácticamente duplica los beneficios obtenidos. **¿Entonces, porque es tan grande el interés en minar criptomonedas en el navegador si es una operación con pérdidas?** Lo interesante de la cuestión el coste computacional lo paga el usuario final por lo que para la persona que inserta el script es un beneficio neto sin el coste asociado a él.

Figura 28 evolución del minado en la web

Desde entonces ha ido proliferando su uso con grandes titulares como cuando se descubrió que ThePirateBay utilizaba este servicio para generar beneficios entre su audiencia, cuando anuncios en YouTube inyectaban mineros en los visitantes o cuando un ataque informático inyectó mineros en las páginas del gobierno de Reino Unido.

Esta técnica es considerada una amenaza siempre que no avise al usuario de que se está produciendo de forma explícita y este de su consentimiento. El problema no es únicamente que el creador del sitio web inyecte el script pertinente para minar a los usuarios del mismo, sino que es simple que existan múltiples entidades que hagan uso de ellas para obtener beneficios.

Muchos sitios web sirven JavaScript de terceras partes, es decir de compañías que no son la misma que ofrece el contenido en el sitio como pueden ser anuncios de una red de anunciantes, herramientas o SDKs de terceros como pueden ser librerías o herramientas de métricas y analíticas. Estas terceras partes tienen los privilegios suficientes para insertar mineros de criptomonedas en los sitios web.

Hay varios casos en que un minero insertado por terceros en una web con un alto tráfico es detectado, un ejemplo seria la web el FCBarcelona., dónde durante algunos días tuvieron corriendo un minero de criptomonedas de la empresa CoinHive. Otros casos conocidos serian cuando web de Movistar se auto inyectó en un test de detección de minado y se usó esa versión en producción por error. La realidad, es que existen miles de sitios web con esta problemática y su resolución no es trivial ya que el objetivo de estos, mayoritariamente, es no ser encontrados y pasar desapercibidos.

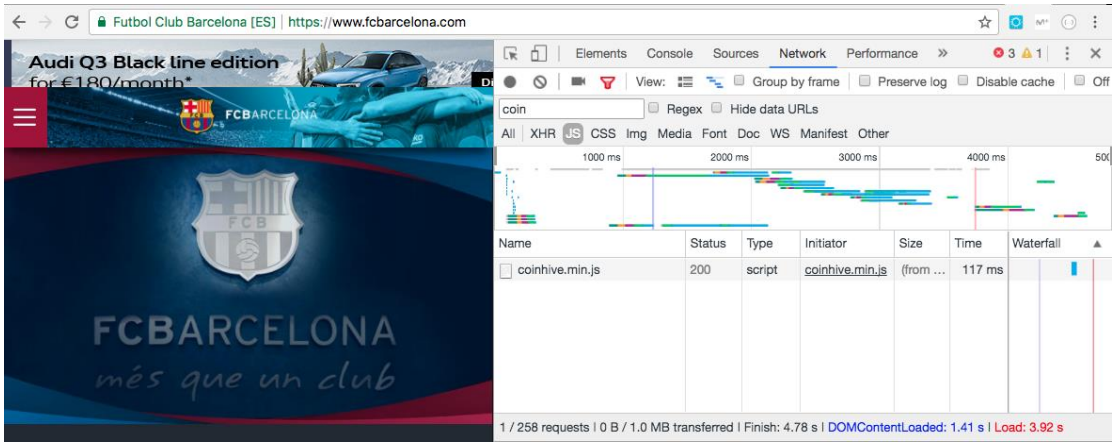


Figura 29 Script de CoinHive insertado de manera ilícita por terceros en la web del FCBarcelona

Otro caso interesante es el que se ha estudiar, es el caso de la inyección de mineros mediante un ataque man-in-the-middle (MITM). Este ataque cuya implementación más usada se conoce como CoffeMiner, consiste en interceptar las respuestas e inyectar en las páginas web solicitadas por el usuario el script de minado por lo que el minado de criptomonedas puede aprovecharse también de las vulnerabilidades de la web.

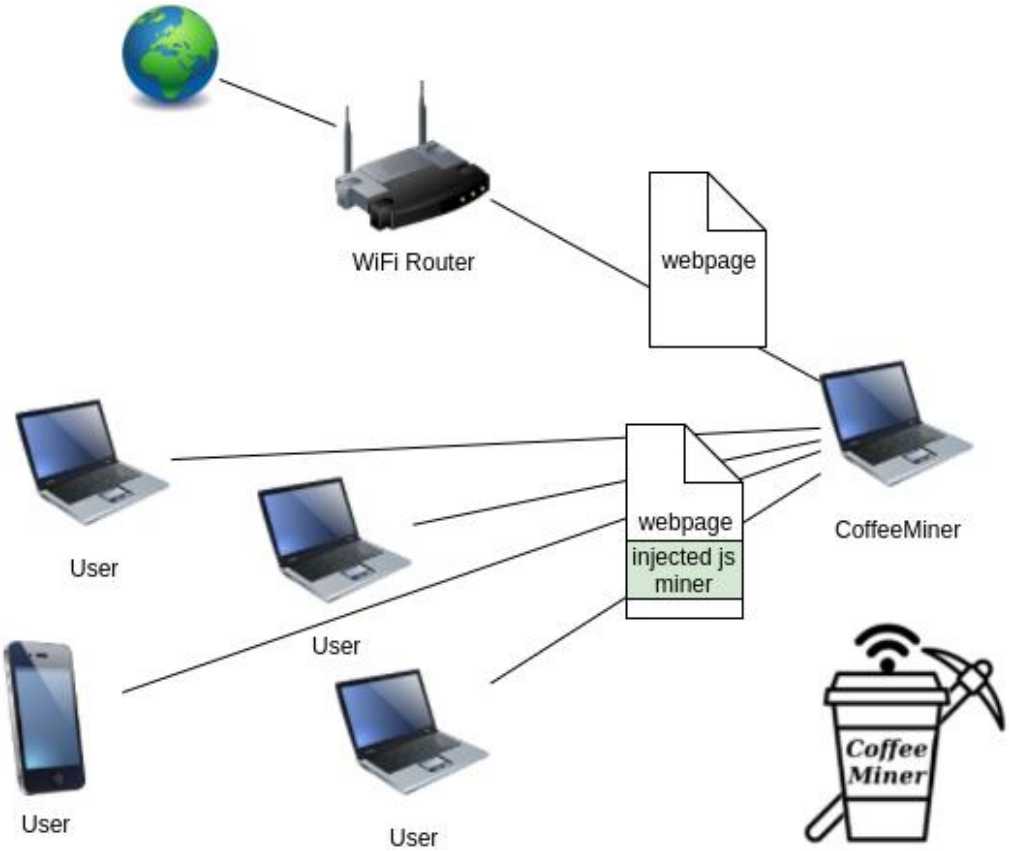


Figura 30 Ataque MITM para insertar un minero

6.3 Detección del minado en la web actual

En este proyecto aparte de analizar el funcionamiento del minado, se ha investigado sobre la posibilidad de detectar y bloquear estas tecnologías que lastran el comportamiento del dispositivo del usuario a la vez que tienen un coste económico para este.

A pesar de ser una técnica bastante novedosa, durante el transcurso de este proyecto he podido observar cómo los tanto los scripts de minado como sus detectores han ido evolucionando en las que he denominado como 4 olas de acción reacción dónde los detectores han aplicado mediadas de detección y los mineros han hallado la manera de sobrepasar las mismas. La figura 29 es muy útil para entender la dificultad que supone seguir identificando las técnicas de ofuscación, ya que a pesar de aumentar el número de sitios usando estas técnicas, cada vez se identifican menos usando las mismas.

Las soluciones actuales están basadas en listas que vamos a ir mencionando en cada uno de los casos, para el análisis se usan las listas de la herramienta OpenSource NoCoin ya que es una forma muy visible de ver como se han ido produciendo los bloqueos.

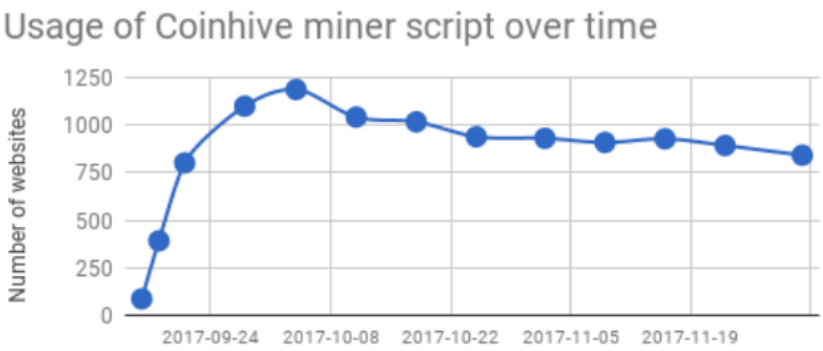


Figura 31 Detecciones por la universidad de Concordia de mineros usando CoinHive

- **Primeros mineros, aún no existe ningún método de bloqueo.**

Las extensiones para navegadores empiezan a reconocer el contenido de los mineros y bloquean los scripts que contengan código conocido para minar. Para ello se hace uso de una herramienta conocida como PublicWWW que es un buscador que indexa todos los códigos fuentes de los sitios web en la red. Es una herramienta muy útil para esta primera fase.

- **Los mineros pasan a ofuscar le código de los scripts mediante distintas técnicas.**

Un ejemplo de ello es el minero utilizado por CoinHive bajo el dominio AuthedMine.com, en el propio script como podemos ver en la figura 30, nos indican que el script ha sido ofuscado para no ser detectado por los bloqueadores de mineros.

Algunos casos de ofuscación interesantes detectados a posteriori de la realización de la primera herramienta de detección por símbolos que no habrían sido posibles de detectar de otra manera.

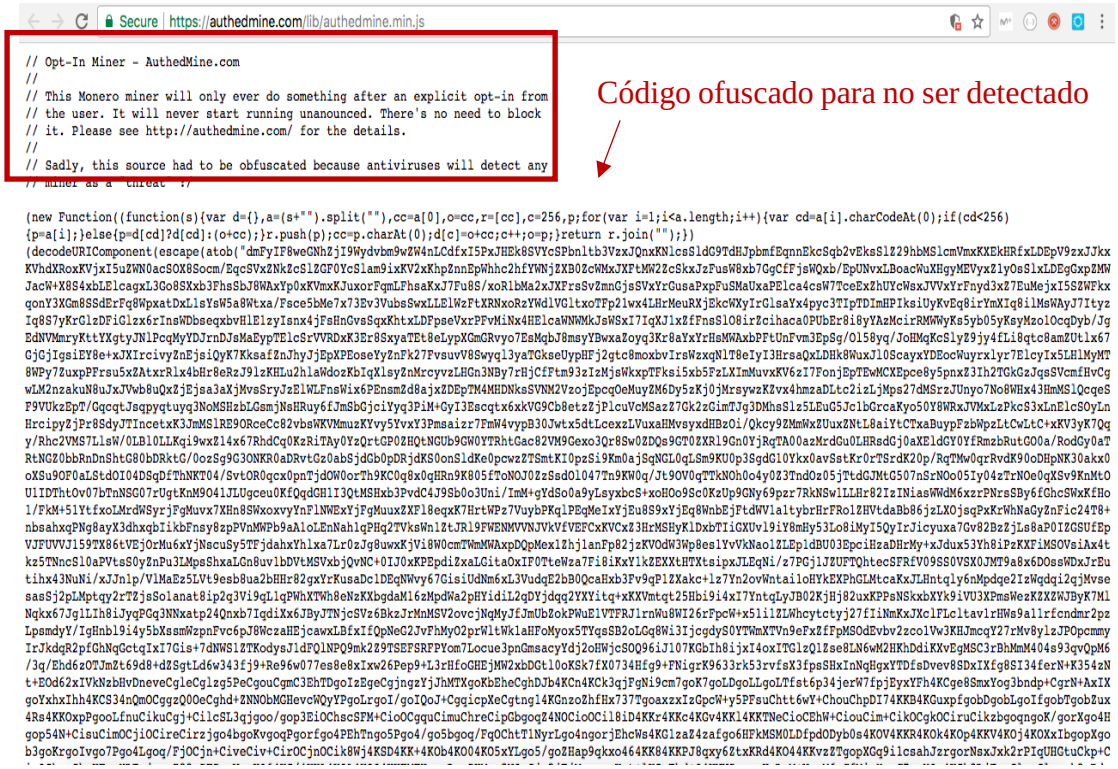


Figura 32 Codigo de authedmine.com el minero de CoinHive

La respuesta de los anti mineros a estas prácticas es bloquear los dominios conocidos para tal fin y los scripts pertinentes. Por ejemplo, la extensión NoCoin pasa a a bloquear los dominios de CoinHive para este servicio en concreto.

```
1  *://coinhive.com/lib*
2  *://coin-hive.com/lib*
3  *://coinhive.com/captcha*
4  *://coin-hive.com/captcha*
```

Figura 33 Dominios bloqueados de CoinHive en la herramienta NoCoin

Los mineros pasan a hospedar los scripts de minado en terceros como amazon webs services, github o en el propio servidor del sitio web.

Un ejemplo de ello es el sitio web mejortorrent.com que como podemos ver en la propia página HTML principal tiene tagueado un script de minado completamente ofuscado e imposible de hallar buscando por nombre o servidor hospedado.

Figura 34 Código fuente de mejortorrent.com dónde se puede ver el script hospedado en el propio sitio

Otros ejemplos hallados a posteriori usando el conjunto de técnicas desarrolladas en los puntos anteriores son las siguientes.

https://akaddn.github.io/lrn-003/m.js → encoding base 64 en string que se carga como función mediante la función de evaluación de código de javascript eval()

http://ofoghnews.ir/wp-includes/js/jquery/jquery.js → código de llamada al script de minado dentro de jquery para ocultar el script de minado que se encuentra en http://178.132.0.217:88/ajax.js que a su misma vez utiliza el nombre Ajax.js para tratar de ocultar el contenido y cuyo código esta codificado en base 64 para ofuscar el uso real.

http://jqcdn03.herokuapp.com/jquery.js → minero que únicamente cambiando el nombre por jquery para ofuscar

Para poder seguir bloqueando los scripts de minado los antimineros pasan a bloquear los WebSockets de las conexiones con los servidores de minado que estos sí que deben ser remotos, como podemos ver en la siguiente imagen la blacklist de NoCoin, el bloqueo de las conexiones a wss://*.coinhive.com/proxy* bloqueará las conexiones a todos los servidores conocidos de coinhive.

```
*://coinhive.com/lib*
*://coin-hive.com/lib*
*://coinhive.com/captcha*
*://coin-hive.com/captcha*
wss://*.coinhive.com/proxy*
wss://*.coin-hive.com/proxy*
```

Figura 35 Bloqueo de los websockets por parte de los bloqueadores de mineros

Como consecuencia, los servicios de minería de datos pasan a usar proxies para establecer las conexiones con sus servidores de tal modo que ya no van a poder ser

detectados siempre que puedan modificar el proxy de minado lo suficientemente rápido para escapar de las listas de bloqueo.

El proyecto más conocido para tal fin es CoinHive Stratum Proxy, que puede ser implementado por cualquiera y únicamente requerirá de inicializar el script de coinhive con la url del proxy en cuestión para que el websocket se establezca con este y no con CoinHive directamente evadiendo así el bloqueo por conexión web socket. Un ejemplo visual es el esquema generado en la siguiente figura.

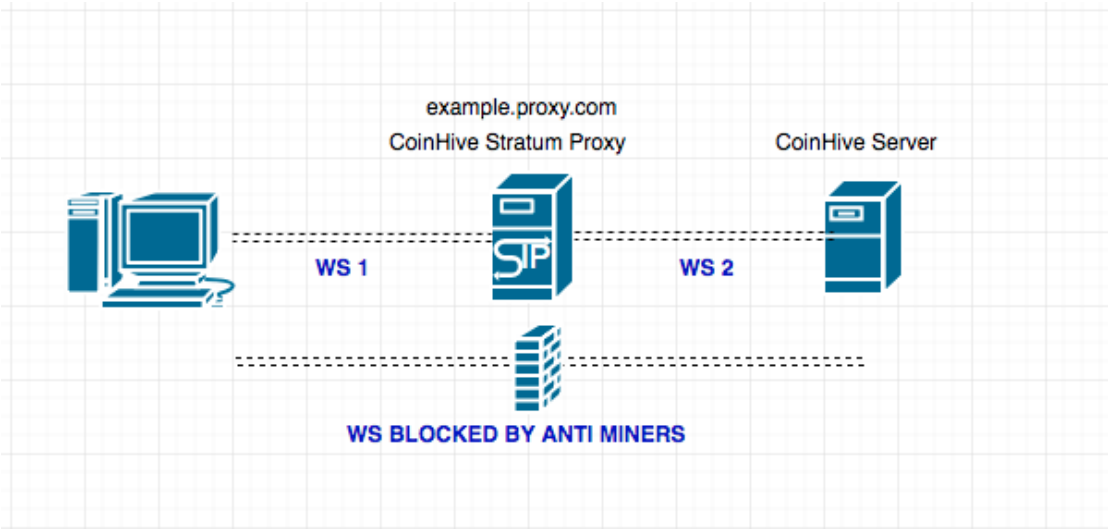


Figura 36 Desmostración funcionamiento coinhive proxy

Otros servicios han optado por cambiar el dominio para saltarse el bloqueo a su dominio principal como es el caso del servicio Mineralt, que tras registrarme a su servicio para poder estudiar su funcionamiento y usar su script en mis tests de detección que voy a explicar en el siguiente apartado, reciví el siguiente correo relatando el cambio de dominio y especificando que el fin era saltarse los adblocks.

> *Dear partner!*
>
> *The domain was changed to improve bypassing Ad block detections. You need to log in your account, copy the new script code and re-install it on your site.*
>
> *Contact us if you have any questions.*
>
> *We are always glad to hear your feedback, suggestions and ideas!*

Y accediendo al panel de control vemos que nos permite añadir proxys de manera automática hacia nuestro propio dominio de manera sencilla.

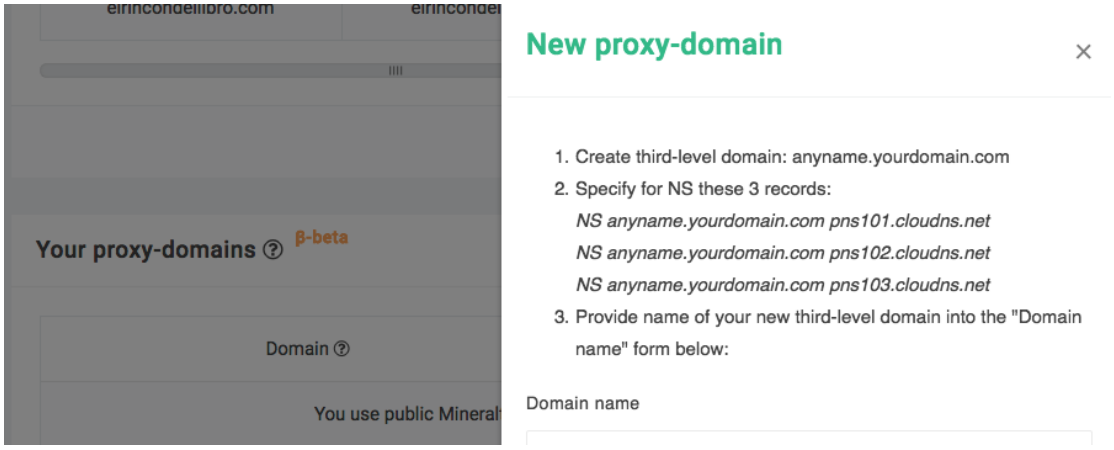


Figura 37 Mineralt nos permite añadir fácilmente proxys

Llegados a este punto, las herramientas de bloqueo por listas ya no son efectivas ya que son completamente evadibles utilizando las tecnologías correctas y claramente el resto de la industria va a evolucionar hacia estas soluciones, con lo que resulta necesario buscar otra estrategia de detección de minado.

6.4 Propuesta de detección mediante análisis de símbolos

La idea que surgió, fue la de utilizar la metodología usada para detectar fingerprinting con el fin de ser capaces de utilizar nuestro web crawler para detectar también minado de monedas. Para ello. Igual que en los casos anteriores, la idea sería conseguir extraer el conjunto de APIs HTML5 utilizadas para realizar el minado y sacar un patrón que nos permita determinar que se está produciendo el minado.

Esta técnica, no ha sido utilizada con anterioridad nunca por lo que partíamos de un riesgo elevado de no ser capaces de hallar un patrón o que el patrón fuera demasiado genérico y encontráramos falsos positivos.

Este método de funcionar correctamente nos permitiría identificar todos los positivos independientemente de todas las técnicas de ofuscación comentadas anteriormente.

La primera idea era ser capaces de identificar un patrón que nos permitiera identificar CriptoNight, el algoritmo de minado de Moneros. El hallar una forma de identificación del algoritmo nos permitiría identificar el global de todos los tipos de minado simultáneamente ya que al final, independientemente de la implementación el Core del algoritmo es CriptoNight.

Para ello, se analizan las implementaciones de CriptoNight en JavaScript. Al analizar varias implementaciones vemos que todas ellas usan un subconjunto de JavaScript, WebAssembly (wasm).

WebAssembly permite utilizar lenguajes como C o C++ para escribir le código que posteriormente será compilado a WebAssembly. Como resultad ose obtiene un código mucho más optimizado que es mucho más rápido tanto en carga como en ejecución.

Para nuestro caso, tiene mucho sentido que sea así dado que es un algoritmo que va a realizar una cantidad de operaciones muy elevadas.

El problema en nuestro caso es que WebAssembly utiliza un subconjunto de JavaScript que no utiliza las APIs de HTML5 para realizar las operaciones ya que estas están mucho más sobrecargadas.

Por poner un ejemplo, para realizar una operación XOR no se utiliza método de Array que nos permita realizar esa operación, sino que se hace uso del operando ^ que va realizar esa operación de manera directa.

Esto supone una gran mejora en rendimiento, pero para el caso estudiado supone un paso atrás en la obtención de una firma para detectar el minado.

El siguiente paso para estudiar la viabilidad de este análisis, es ver si podemos sacar alguna conclusión en referencia al conjunto de operaciones alrededor del algoritmo de generación de hashes, es decir, ver si de alguna manera podemos determinar un conjunto de operaciones que se realizan previa generación del hash y a posteriori.

Para ello se analizan los scripts de CoinHive y el resto de empresas del sector más relevantes y se obtiene la siguiente tabla.

Call	coinhive	mineralt	coinimp	coinhave	minerpw	monerise	nahnoji
BroadcastChannel.postmessage	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
BroadcastChannel.onmessage	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE
navigator.hardwareConcurrency	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
XMLHttpRequest.addEventListener	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
XMLHttpRequest.open	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
XMLHttpRequest.send	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
XMLHttpRequest.responseText	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
XMLHttpRequest.responseType	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
XMLHttpRequest.ontimeout	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
XMLHttpRequest.onerror	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
XMLHttpRequest.status	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
Worker.onmessage	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
Worker.onerror	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
Worker.postMessage	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
WebSocket.binaryType	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE
WebSocket.onopen	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
WebSocket.onclose	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE

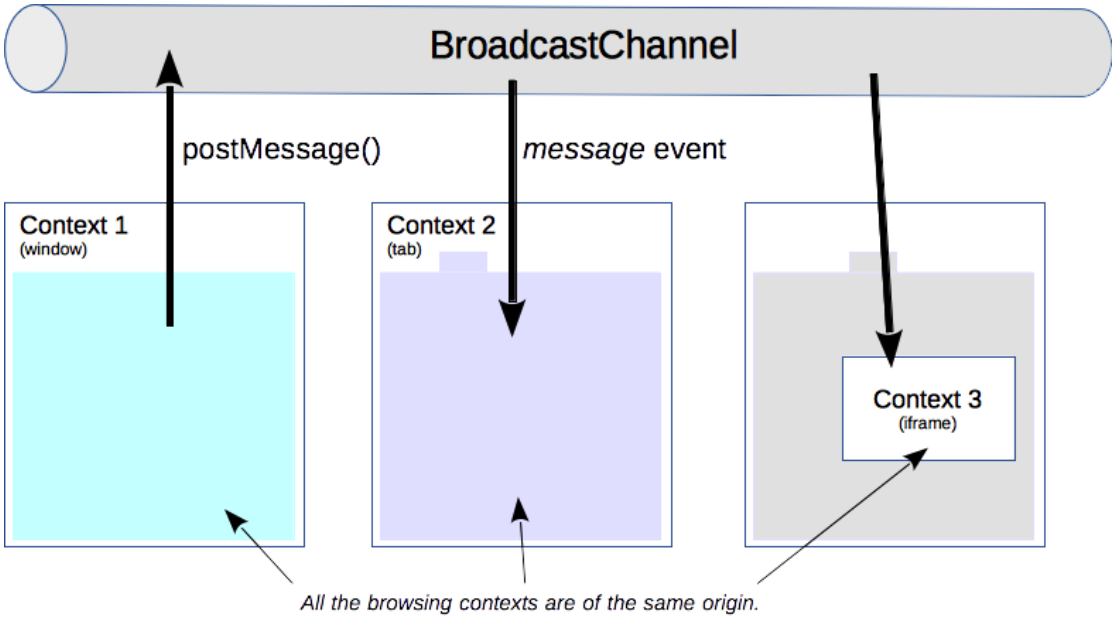
WebSocket.onerror	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
WebSocket.onmessage	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
WebSocket.send	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
WebSocket.protocol	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Tabla 4 Comparativa APIs HTML5 utilizadas por los diferentes servicios de minado

Para realizar un análisis más concreto vamos a centrarnos en el caso más usado que sería CoinHive.

BroadcastChannel.postmessage & BroadcastChannel.onmessage:

Las llamadas a la API BroadcastChannel son necesarias para comunicarse con los hilos de ejecución que van a realizar el computo del hash (Workers), esta API nos permite desde diferentes contextos escuchar a eventos de nuevos mensajes y escribir nuevos en el bus de BroadcastChannel. Veremos cómo esta API es imprescindible para sacar el máximo rendimiento del script de minado ya que de otra manera únicamente podríamos tener un hilo de ejecución.



navigator.hardwareConcurrency:
Esta propiedad nos devuelve el número de procesadores lógicos disponibles, o l oque es lo mismo, el número de threads o hilos de ejecución que es capaz de lanzar el navegador. Esta propiedad de la variable navegador, es muy útil para lanzar en cada hilo un worker y sacar máximo el rendimiento del minero. Aunque es usada por todos los navegadores, algunos han empezado a prepararse para la posibilidad de que esta API sea bloqueada dado que invade en cierto modo la privacidad del usuario.

Worker:

Los Workers son utilizados para ejecutar código JavaScript en background. De tal manera que no afectan a la ejecución del sitio web bloqueando la misma dado que JavaScript es monothread. Lo que se hace es crear tantos Workers como hilos de ejecución estén disponibles, de esta manera maximizamos el número de hash que se obtienen. La comunicación se realiza mediante el bus de BroadcastChannel.

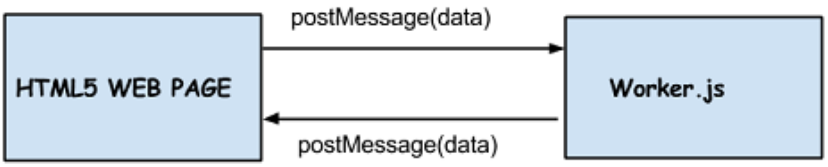


Tabla 5 Comunicación de la página principal con los Workers

WebSocket:

Los WebSockets permiten establecer sesiones de comunicación interactiva entre un cliente y un servidor. De tal manera que no tenemos que interrogar al servidor para recibir respuestas de este, sino que podemos esperar a recibir eventos.

El siguiente ejemplo de un minero de CoinHive muestra lo comunicación mediante un WebSocket, dónde el cliente manda su identificador y a posteriori vemos que re reciben dos mensajes del usuario, el primero para notificar que ha sido validado correctamente al usuario y que aún no ha completado ningún hash.

A posteriori se recibe otro mensaje dónde se recibe un blob. Que básicamente es un encapsulamiento para enviar raw data en binario. Posteriormente se resuelve el hash de ese blow y servidor nos confirma los hashes que han sido aceptados. El proceso se va repitiendo hasta que se interrumpe la ejecución del script.

Data	Length	Time
↑ {"type":"auth","params":{"version":7,"site_key":"E1BjdrtuctlEiM5WshwYFb6CtP…	124	10:41:54.868
↓ {"type":"authed","params":{"token":"","hashes":0}}	50	10:41:55.126
↓ {"type":"job","params":{"job_id":"566417344356887","blob":"07079280afd8059…	234	10:41:55.131
↑ {"type":"submit","params":{"version":7,"job_id":"566417344356887","nonce":"9…	162	10:42:21.888
↓ {"type":"hash_accepted","params":{"hashes":256}}	48	10:42:21.950
↑ {"type":"submit","params":{"version":7,"job_id":"566417344356887","nonce":"3…	162	10:42:52.607

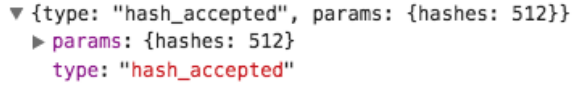


Figura 38 WebSocket de un minero

Con la información recopilada se procede a añadir las APIs HTML5 analizadas a nuestra herramienta de detección de tal modo que estas APIs sean detectadas y sus llamadas grabadas en nuestra base de datos para su posterior análisis.

```
643      // Access to minning
644      instrumentObject(window.Worker.prototype,"Worker");
645      instrumentObject(window.WebSocket.prototype,"WebSocket");
646      instrumentObject(window.BroadcastChannel.prototype,"BroadcastChannel");
647      instrumentObject(window.XMLHttpRequest.prototype,"XMLHttpRequest");
648      instrumentObject(window.URL.prototype,"URL");
649      instrumentObject(window.navigator,"navigator");
650
651      instrumentObject(window.Uint8Array.prototype,"Uint8Array");
652      instrumentObject(window.TextDecoder.prototype,"TextDecoder");
653      instrumentObject(window.Int16Array.prototype,"Int16Array");
654      instrumentObject(window.Int8Array.prototype,"Int8Array");
655      instrumentObject(window.Int32Array.prototype,"Int32Array");
656      instrumentObject(window.WebAssembly.Memory.prototype,"WebAssembly.Memory");
657      instrumentObject(window.WebAssembly.Table.prototype,"WebAssembly.Table");
658      instrumentObject(window.Float32Array.prototype,"Float32Array");
659      //instrumentObject(window.ArrayBuffer.prototype,"ArrayBuffer");
660      //instrumentObject(window.Array.prototype,"Array");
661
662
663      console.log("Successfully started all instrumentation.");
```

Figura 39 Resgistro de todas las APIs que queremos analizar

En el script de análisis de igual manera que anteriormente para fingerprinting buscamos la aparición de esos símbolos en un mismo sitio web, en un orden determinado que concuerde con el funcionamiento de fingerprinting deseado.

```
34  print "\n ------------------------------------------------------------"
35  print " -------------- Cryptocoin miners detection ------------------"
36  print " ------------------------------------------------------------\n"
37
38
39
40  cur.execute(("SELECT DISTINCT top_url, script_url FROM javascript_view "
41               "WHERE symbol = 'BroadcastChannel.onmessage';"))
42  BroadcastChannel_onmessage = set()
43
44  for top_url, script_url in cur.fetchall():
45    try:
46      BroadcastChannel_onmessage.add((top_url, script_url.split(' ')[0]))
47    except:
48      pass
49
50  cur.execute(("SELECT DISTINCT top_url, script_url FROM javascript_view "
51               "WHERE symbol = 'navigator.hardwareConcurrency';"))
52  navigator_hardwareConcurrency = set()
```

En este trozo de código del script de análisis de minado mediante CoinHive realizado en Python, podemos ver como se generan unos sets con los positivos para cada símbolo. Finalmente se busca la intersección ordenada temporalmente de todos los elementos que hemos visto que conforman la técnica de minado y de allí se extraen los positivos,

que van a consistir en unas tuplas con la URL del script y la URL del sitio web que van a ser insertados en nuestra base de datos MonogDB.

6.5 Análisis de resultados

Para poder probar el correcto funcionamiento de la herramienta tal y como se había planteado, se genera un servidor con varios de los servicios comentados hospedados en él. Básicamente se genera una página web para cada tipo de minado y se sube a nuestro servidor público para poder ser accedido por nuestra herramienta de Crawling y observar si somos capaces de detectar los positivos.

```
Albert-MAC:cryptotests atm$ ls
404.html                    firebase.json               mineralt.html
coinhave.html               index.html                  mineralt_jquery.html
coinhive.html               index.html.firebase.html    minerpw.html
coinhivemanager.html        invemo.html                 monerise.html
coinimp.html                jquery.min.cryptominer.js   nahnoji.html
```

Aparte de los test creados, con los scripts de minado de los diferentes servicios, se han añadido algunos sitios conocidos por usar mineros de tal modo que se obtiene una visión de su funcionamiento en webs reales por su hubiera algún tipo de variación.

```
openwpm@dev-anda-dtl-01:~/privacymeter/utils/generate_lists/topsites$ cat crawl_lists_test_miner
s_mongo_ok/test_miner_mongo
http://batmanstream.com
http://deccanchronicle.com
http://shareae.com
http://ilcorsaronero.info
http://porjati.ru
http://oipeirates.se
http://khbrbgd.com
http://eduvision.edu.pk
```

Tras realizar este test varias veces y ajustar algunos de los servicios para ser detectados inequívocamente entre ellos, tenemos la certeza de poder identificar los servicios en la red y la única variación que deberemos tener en cuenta son los posibles falsos positivos que analizaremos una vez tengamos el test general de los 400.000 sitios web.

Después del primer test podemos concluir que no hay falsos positivos después de analizar los 570 positivos hallados. Por lo que podemos sacar tres grandes conclusiones:

1. Es el primer método de detección automatizado capaz de asegurar la detección a pesar del uso de técnicas de ofuscación.
2. Es un método fiable ya que de 400.000 sitios web no hemos obtenido ningún falso positivo.
3. En un futuro análisis tenemos que ver alguna de las técnicas utilizadas nos pasa desapercibida y no es detectada.

7. Análisis de los resultados de la herramienta de Crawling

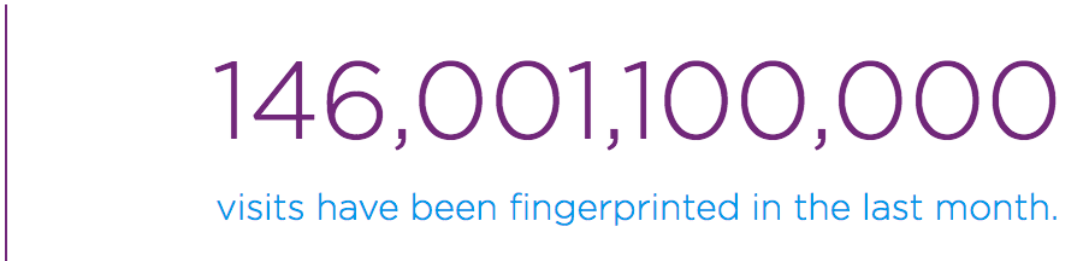
En este apartado se proceden a estudiar los resultados obtenidos por la herramienta de Crawling implementada para las dos técnicas estudiadas.

7.1 Resultados del análisis de fingerpriting

La herramienta de Crawling se ejecuta mensualmente, y se publican los datos mensualmente. En total se analizan los 400.000 sitios web más visitados según el ranking Alexa, que junto a nuestra base de datos de información del top 1 millón de sitios más visitados, podemos complementar los datos obtenidos y sacar conclusiones interesantes, que debido a que nadie a ha realizado la tarea de complementación antes.

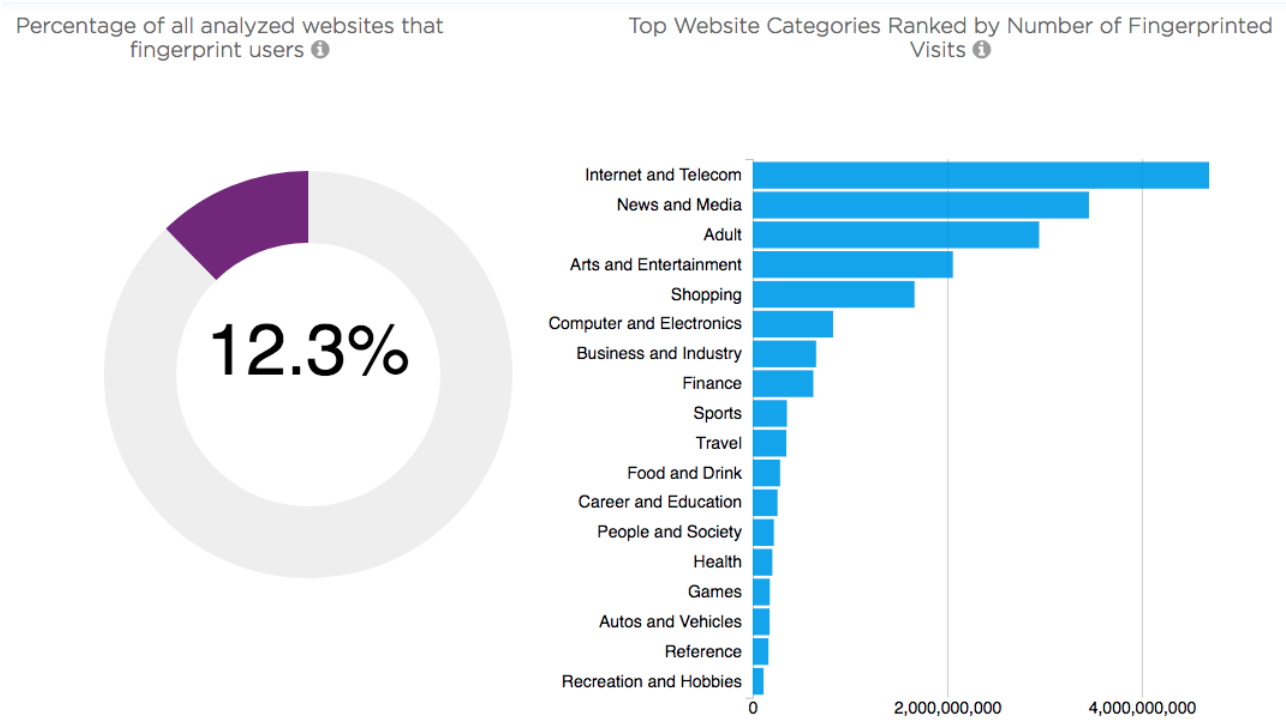
Del análisis podemos sacar los siguientes datos.

Analizando todos los positivos y el tráfico que tenemos de cada sitio de ellos vemos que un total del de 146.001.100.000 visitas han sido fingerprinteadas en el último mes. Unos datos que nadie había mostrado antes y menos de manera actualizada mensualmente.

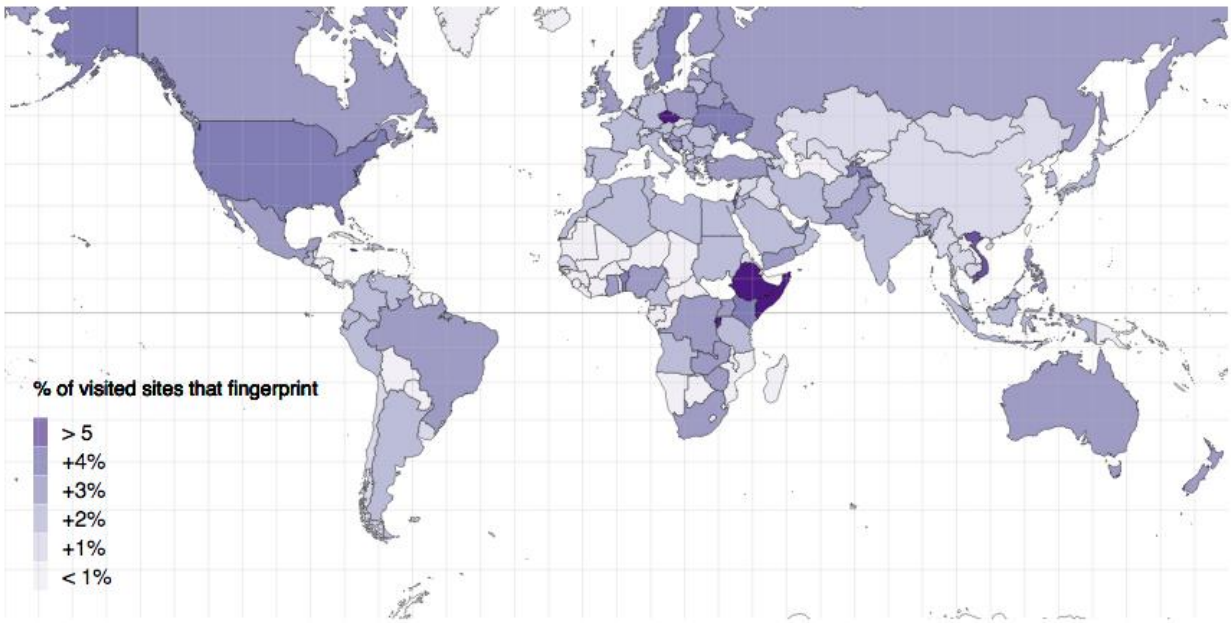


Otros datos interesantes, es observar en que porcentaje de sitios web se han encontrado scripts de fingerprinting que como podemos ver en el gráfico circular es un total del 12,3%. Es un dato muy curioso ya que puede parecer una técnica poco usada pero que realmente tiene un impacto real en la industria de la identificación del usuario.

En el segundo gráfico, podemos ver que sectores son los que registran más visitas con fingerprinting. Es interesante observar como las categorías con más visitas fingerprinteadas son las que en general mueven más dinero y dónde los anuncios tarjeteados son más usados. Si nos fijamos en el top 5 de categorías, podemos ver categorías como Compras, Noticias, Adultos, Internet (buscadores etc.). Vemos que esta es una tendencia que se preserva de test a test mensualmente.



Otro análisis que se ha realizado es el porcentaje de sitios que usan fingerprinting en las webs más visitadas de cada país, esto nos va a permitir sacar conclusiones de que influencia tiene, a pesar de no ser una práctica explícitamente regulada, las regulaciones más estrictas en el uso de las técnicas de fingerprinting.

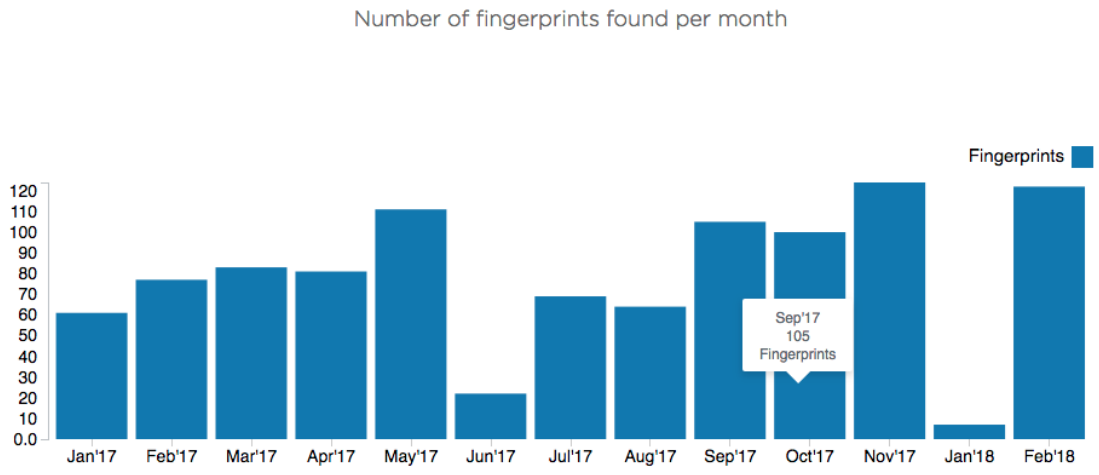


Como vemos en el mapa, en general Europa representa un porcentaje de fingerprinting inferior comparado con por ejemplo Estados Unidos, es interesante ver como por ejemplo en China, el nivel de fingerprinting es inferior a la mayoría de países desarrollados, esto es debido mayoritariamente a que China tiene sus propias redes de

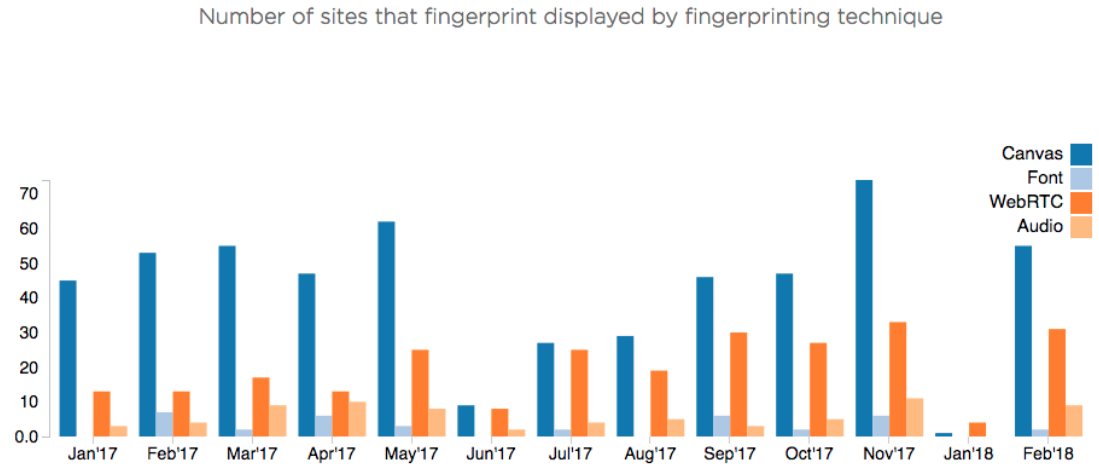
distribución de anuncios que al final son la fuente principal de interesados en obtener estos datos y como consecuencia los que inyectan los scripts que realizan fingerprinting.

Un caso muy interesante es Vietnam, con una tasa de fingerprinting muy por encima del resto, analizando los resultados, vemos que igual que China, Vietnam tiene un servicio de anuncios propio (admicro.vn usando el script media1.admicro.vn/core/fipmin.js), el cual hace uso de técnicas de fingerprintig. Por lo que un porcentaje muy alto de los sitios nacionales contienen scripts de fingerprinting.

Otros datos interesantes son por ejemplo el número de fingerprints en el top 1000 analizado mes a mes. Como vemos en ambos gráficos en Junio de 2017 y Enero de 2018 tuvimos un problema con el acceso a las dns desde el servidor y los resultados son corruptos y no válidos.



Como podemos ver, en el top 1000, los datos son bastante constantes, podemos ver que hay una tendencia alcista, aunque no excesiva.



En este segundo gráfico vemos los 4 tipos de fingerprinting que se analizaron desde el primer test (WebGL se registra pero al no tener datos relevantes aún no se muestran en el gráfico), podemos ver que la técnica de fingerprinting más usada es Canvas, esto es

debido a que es la técnica que hemos podido observar que genera más entropía y más sencilla de implementar. Por otro lado, audio fingerprint al ser una técnica bastante novedosa vemos que, aunque a un ritmo lento, su uso va creciendo.

7.2 Resultados del análisis de criptominado

Para el análisis del uso de minado de criptomonedas al ser una técnica que se implementó mucho más tarde, únicamente se tienen los resultados de un test de criptominado completo. Por lo que los resultados se adjuntan en tabla.

TOP	TOTAL (#,%)		FROM COINHIVE (#/%)		"OBFUSCATED" (#/%)	
10000	15	0.150%	11	73%	4	27%
100000	155	0.155%	94	61%	61	39%
400000	570	0.143%	350	61%	220	39%

Tabla 6 Resultados del test de criptominado

Como podemos ver en la tabla de resultados, el uso de minado de criptomonedas en el navegador es aún minoritario, estando presente en alrededor del 0.15% de los sitios web, en total en 570 sitios se encontró un minero.

Es interesante ver que el 73% de los positivos usan CoinHive, el servicio más utilizado con diferencia y que el resto de scripts provenían de scripts enmascarados que no provenían de un servicio de minado en concreto.

Este último caso ya ha sido solventando añadiendo los cinco principales servicios de minado de criptomonedas en la red y, que a pesar de que sean ofuscados, seremos capaces de detectar debido a la naturaleza de nuestro sistema de detección.

Adult	Arts*	Business*	Career*	Computer*	Internet*	News*	OTROS
0	5	0	0	0	1	0	9
17	32	1	4	6	11	9	75
41	84	28	16	27	40	26	308

Tabla 7 Categorías que hacen más uso de criptominado

Si nos fijamos en esta tabla de uso de criptominado en las categorías dónde más apariciones tenemos, vemos que Arte es la categoría dónde más se utiliza. Este dato podría parecer extraño, pero no lo es dado que El ranking Alexa categoriza cualquier tipo de contenidos bajo la categoría arte. Es una estadística con mucho sentido dado que el minado de criptomonedas, cuanto más tiempo se esté ejecutando, más beneficios reportará, por lo que sitios de streaming o descarga dónde el usuario va a estar un tiempo mucho mayor en el sitio, va a ser un sistema de monetización muy rentable. De la misma manera es extrapolable a las categorías de Adultos o Internet.

8. Herramientas de visualización de resultados

Desde el inicio el planteamiento de este proyecto ha sido dar visibilidad a los resultados obtenidos de una manera clara que sea útil para los usuarios, reguladores en materia de privacidad y periodistas que puedan con estas herramientas tener una mejor idea de las técnicas usadas para rastrear a los usuarios. Para ello se plantearon varias opciones de visualización y se decidió apostar por dos en general, una versión web de los resultados y un plugin/extensión para Chrome que mediante el uso de nuestra API informe al visitar un sitio que use fingerprint y dando información al usuario sobre las técnicas.

8.1 Web

La idea del diseño del sitio web siempre ha sido generar un diseño claro, visual, sencillo que ofrezca la información de los resultados a los usuarios de manera directa y entendible. Al ser un producto final que tiene que ser visible por el público y presentado como proyecto de manera interna en telefónica, no pude dar la sensación de prototipo, se ha realizado este sitio web no a modo de demostración sino como una web terminada. Para ello se han considerado las mejores tecnologías para conseguir una web rápida, y que pudiera ser hosteada en un servidor gratuito ya sea de Github (github.io) o el servicio que ofrece Firebase de hosting. Aunque la primera versión se subió a Github, por la posibilidad e modificar código usando git directamente, para la versión final se ha decidido que la mejor opción es usar Firebase por dos motivos, al tener la base de datos y el sitio web en el mismo servicio conseguiremos la máxima velocidad y segundo, más importante, el hecho de que Firebase haga uso de un modelo distribuido mediante CDN, hará los usuarios de todo el mundo reciban el sitio web del servidor de Firebase más próximo a ellos, optimizando de este modo también la velocidad.

Para realizar el sitio web, se han usado varias tecnologías diferentes, se ha usado Jekyll como parser empaquetado en una gem de Ruby, este parser es una opción muy interesante ya que nos genera contenido estático construido el contenido. Es compatible con todo tipo de tecnologías como JavaScript o estilos CSS por lo que no tenemos ninguna desventaja en ese sentido y no hay que realizar el contenido de manera diferente. Las ventajas que nos ofrece Jekyll es su versatilidad ya que podemos crear la página como si de una página HTML se tratara, añadiendo nuestros frameworks y código sin problemas. Seguridad ante ataque, al no usar ningún tipo de base de datos y ser la página final puramente contenido estático, lo hace robusto frente a ataques y mejor opción que por ejemplo Sinatra o Rails. El hecho de que el contenido sea estático, también da mucha velocidad a su distribución, cosa que en nuestro caso ese muy interesante. Finalmente, como ya hemos comentado previamente, es posible alojar un sitio web creado con Jekyll en servicios gratuitos y de calidad como Github pages o Firebase Hosting.

Su uso es muy directo,

```
gem install Jekyll // instalamos la gem de Jelyll

jekyll new my-new-site // creamos nuestro sitio web

cd my-new-site // nos situamos en el directorio dónde encontramos
```

```
jekyll serve //se genera el contenido y se ejecuta el servidor
```

Otra ventaja que nos ofrece Jekyll es la posibilidad de crear el código de manera fraccionada, realizando las diferentes partes de cada documento HTML en diferentes documentos con lo que a posteriori podemos reutilizar un header o un footer sin reescribir el código a requerir de frameworks más complejos como angular.

Para el desarrollo del sitio, se ha optado por el uso de HTML y los estilos proporcionados por bootstrap, que nos ayudan a obtener diseño depurado y responsive que se adapte a todos los dispositivos. Para el diseño del sitio web se ha elegido el diseño single page website, donde toda la información está disponible en la página principal. Eso no es del todo cierto ya que a posteriori se han implementado opciones que se ha visto que debían ser incorporadas en una nueva página cómo es el buscador de un sitio web concreto.

La página web tiene el objetivo de difundir los resultados obtenidos, con tal objetivo en mente se han descrito las técnicas estudiadas, en este caso y para esta primera versión de Web Privacymeter, las técnicas de fingerprinting, versión de manera muy simplificada a modo informativo.

Posteriormente, se muestran los diferentes resultados que hemos concluido como interesantes. Los datos se obtienen de Firebase directamente con lo que siempre están actualizados. Con el objetivo principal de difusión, se decidió que se debía generar un sitio web muy visual. Por ello se ha hecho uso de la librería D3.js que nos ofrece muchas posibilidades de visualización. D3 es una librería para la manipulación de DOMs, es decir, la manipulación de datos estructurados. D3 nos permite la manipulación de esos datos y la posibilidad de su posterior representación en gráficos generados o usando modelos ya existentes. En nuestro caso se han adaptado soluciones estandarizadas a las necesidades del proyecto.

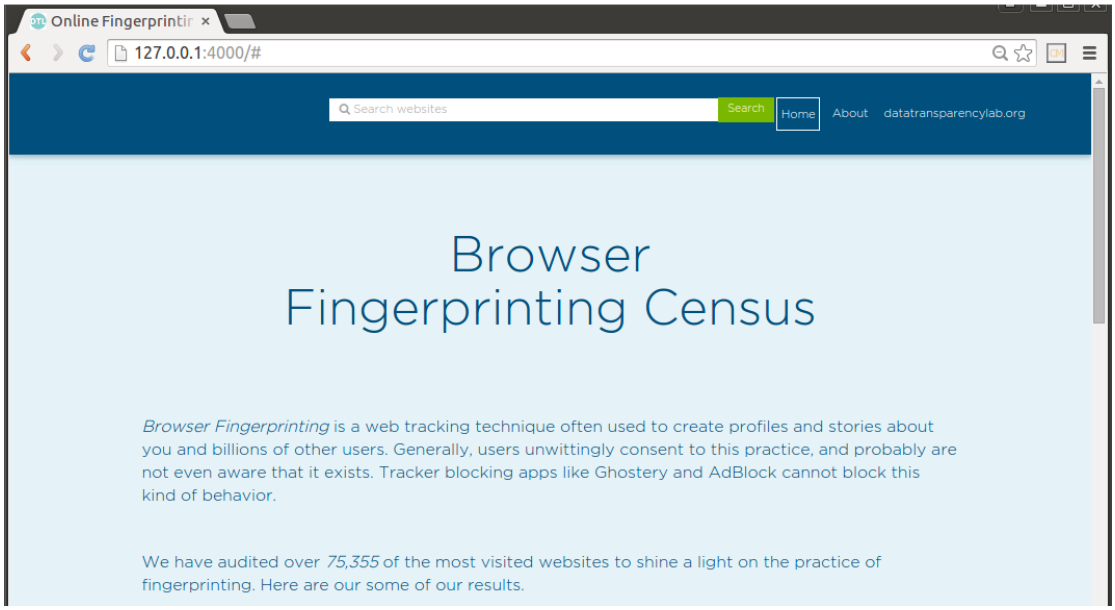


Figura 40 Diseño de la web de resultados

En esta primera página se puede apreciar la descripción, de las técnicas de fingerprinting a modo de difusión. La estructura como se ha comentado previamente ha sido creada con Jekyll con lo que, aunque una vez parseado se ejecuta como una única página stand-alone, realmente son varios ficheros que componen uno de la siguiente manera:

```
--------------------------index.html-------------------------------


---
layout: default
isIndex: true
title: Data Transparency Lab
---

CONTENT

-------------------------default.html----------------------------

//estructura general

<!DOCTYPE html>
<html xmlns:ng="http://angularjs.org" ng-app="fingerprinting">

          {% include head.html %}

          {% include header.html %}

          {{ content }}

          {% include footer.html %}

  </body>


</html>
```

Como vemos se hace uso del layout default por lo que se ejecuta la estructura que se encuentra en la página default.html. De este modo podemos aprovechar el header y el footer, en nuestro caso, para cualquier página que deseemos. Estos dos documentos, van a ser reutilizados para la página de informa para un sitio web concreto mediante el buscador.
En las siguientes capturas podemos ver las diferentes visualizaciones que se han realizado, como se comenta anteriormente se han modificado varias visualizaciones implementadas sobre la librería D3, D3 nos permite leer el contenido de un JSON, en nuestro caso los que encontramos en la base de datos para posteriormente dibujar con los datos de este los gráficos siguientes. Esta librería es sobre todo una librería de manipulación de datos (DOM) y no de dibujo de gráficos. Para ello hace uso de formatos como SVG que permiten usar imágenes vectorizadas.

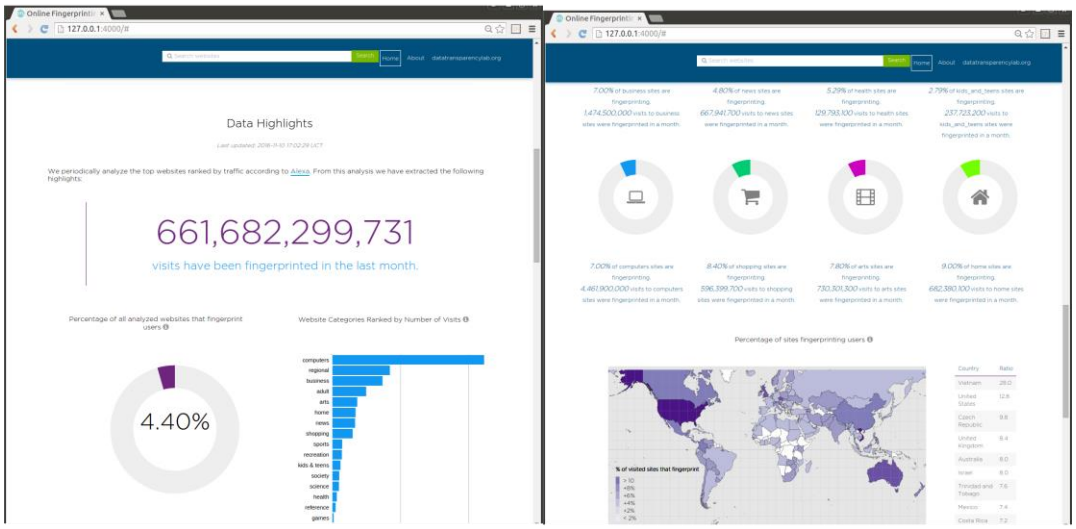


Figura 41 Gráficos generados con la librería D3

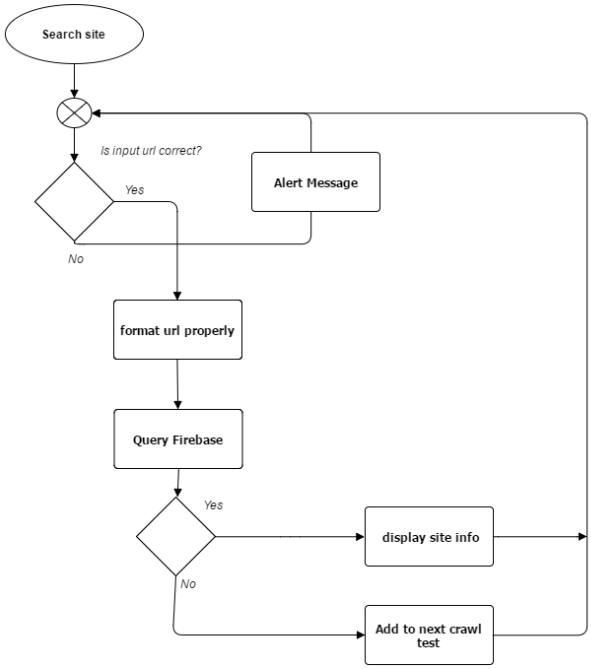
Como podemos ver, se han manipulado los gráficos para ser adaptados a nuestras necesidades, el gráfico del mapa por ejemplo ha sido elaborado desde 0 mediante un mapa vectorizado, estos mapas son de libro acceso y son llamados topojson, la posterior manipulación ha sido realizada íntegramente para poder generar los colores en función del número de fingerprints.

El sitio web también permite obtener los detalles para un sito concreto, en la siguiente imagen se muestra un caso positivo de fingerprint como sería la web de Dropbox. Para este documento se ha decidido mostrar el contenido en consola ya que muestra más información de la interacción que se ejecuta. El proceso de búsqueda que se ha implementado es el siguiente:

- Comprobar que la URL introducida es válida.
- En caso afirmativo, codificar en base16 la url que en nuestro modelo de datos será el identificador del sitio web.
- Se lanza query a la base de datos por ese identificador.
- En caso afirmativo se devuelve un objeto JSON con el resultado, si ese sitio no ha sido analizado, se añade a la base de datos para ser añadido en el próximo test.

En la siguiente página se puede observar el diagrama de los diferentes estados de esta búsqueda. Estas funcionalidades han sido implementadas usando JavaScript, creando, por ejemplo, para codificar base 16, o usando funciones como las propias de la API de Firebase para interactuar con la base de datos.

Web Privacymeter Search bar Interaction



8.2 API

Para acceder a los resultados se ha generado una API REST en Firebase, se definen una serie de funciones las cuales se pueden acceder mediante los métodos HTTP GET y POST. Al hacer uso de HTTP se hace posible el uso del servicio desde prácticamente cualquier lenguaje, lo que es imprescindible en un proyecto como este donde la intención es que esa API sea usado por terceros. Esta API ha sido perfectamente detallada en inglés para abarcar todos los posibles desarrolladores interesados, la documentación de la API se adjunta a continuación ya que es una parte imprescindible del proyecto dada la naturaleza de difusión de este, también es accesible en el repositorio del proyecto. Se ha adjuntado en este documento ya que es auto explicativo y permite entender el funcionamiento de la API

What is Privacymeter?

Privacymeter is a tool build over the opensource framework openWPM which is expected to quantify privacy found in the web, for each site analysed, give an idea of how secure or insecure a site is by analysing different parameters (at this moment fingerprint techniques) so people can have an idea of how they are being tracked, by who and also how dangerous it could be. This results should be available to build other tools over it.

How is the API built?

This is API is built using Firebase, this means that all the data is available in a Firebase tree structure and can be queried by either requesting the top-level branches (to get the whole set of information) or sub-branches to get following the tree branches. The responses are always JSON files that describe the tree structure corresponding to the URL that has been queried (e.g. the endpoint that has been requested)

Firebase

Firebase is a service that let developers store information in tree structure that can be easily retrieved by developers, especially in Web environments, as the whole structure or sub-structures can be retrieved as json objects that can be easily manipulated. This means that by storing all the information in Firebase we get automatically an API to fetch data from it.

Firebase SDK

You can query data hosted in the Firebase either by directly requesting the endpoint URLs as explained below or using any of the Firebase SDKs: Web, iOS, Android or REST.

The root endpoint

The information is stored in a tree, that can be visualized at the Root URL of the Firebase. The URL of the Privacymeter endpoint is: https://privacymeter-dtl.firebaseio.com/ .

Sub-URLs can be also used to show only part of the tree in a browser (e.g. https://privacymeter-dtl.firebaseio.com/insigths) or to get directly the JSON representation of that part of the tree (e.g. https://privacymeter-dtl.firebaseio.com/insigths.json)

Return format

The API will return the structure of the tree serialized as a JSON object.

Passing Parameters

Additional parameters can be passed by adding them to the root URL endpoint

Some examples:

Getting the results insights

In this case we just need to append the URL of the branch that contains all the apps (it's important to use the .json prefix to get the json representation and not the graphical one):

https://privacymeter-dtl.firebaseio.com/insigths

Getting information of certain analysed site

In this case we deactivated the bulk sites download so, https://privacymeter-dtl.firebaseio.com/sites.json won't work and will return a "error" : "Permission denied" message. But childs are possible to query using the following format:

ex. to query google.com, we need to encode it to 16 bit lower case: 676f6f676c652e636f6d.

https://privacymeter-dtl.firebaseio.com/sites/676f6f676c652e636f6d.json, being it the correct query and will return us the info.

If result is null, it means it doesn't appear in our database, you can submit a query to add that website to our test.

In case we want to know the information of the scripts performing more fingerprints, we need the the scripts branch like in the following example:

https://privacymeter-dtl.firebaseio.com/scripts.json

Ask to test a certain website

If we just want to know the information about one website that is not available, what you can do is just using the endpoint where the information of that application is stored and post (example using curl):

curl -X POST -d '{"url": "google.com"}' 'https://privacymeter-dtl.firebaseio.com/queries.json'

USAGE IDEAS

Interactive Visualizations

One potential usage of this API is providing an interactive visualization about how websites are leaking information, which type of information and to which domains. A draft visualization is already available at the DTL GitHub page.

Browser Plugin

Another potential usage is developing a Browser plugin that when browsing the web, provides information about the potential information leakages of that current website.

Also, the plugin could be used decide new websites to analyse.

How much does your site leak?

Another alternative would be a website informing website owners what information their site is leaking.

API LIMITATIONS

Right now our API supports around 100 simultaneous connections. If needed it can be increased.

FEEDBACK

Both the project and the API are in a very preliminary status. Hence, we would be grateful about any feedback you can give us!

Types of feedback we are eager to get:

Is any missing information that could make this data more useful?
Is there are any better way to structure the information?
Is there any missing API?

CONTACTING US

Join #datatransparencylab on irc.freenode.net to chat with us real time, or leave a message that will be logged

SOME USEFUL LINKS

Privacymeter Project

https://datatransparencylab.github.io/fingerprinting-census/

Data Transparency Lab

http://www.datatransparencylab.org/

Firebase SDK

https://firebase.google.com/docs/

Sample Visualizations

https://datatransparencylab.github.io/fingerprinting-census/

Usage Examples

https://privacymeter-dtl.firebaseio.com/insights.json
https://privacymeter-dtl.firebaseio.com/sites/676f6f676c652e636f6d.json
https://privacymeter-dtl.firebaseio.com/scripts.json

9. Herramienta de detección en tiempo real

9.1 Objetivo

El objetivo de esta herramienta es ser capaces de detectar en tiempo real la ejecución de técnicas de fingerprinting y minado de igual manera que hemos realizado en la herramienta de Crawling.

Aunque puede parecer que únicamente va a consistir en una migración de código, la implementación de ambas es muy distinta debido al permiso de acceso del que dispone la extensión los cuales por motivos obvios son mucho más limitantes que los que pueda tener una infraestructura de investigación debido a las implicaciones en cuanto a privacidad del usuario que existen en un navegador comercial.

De hecho, aunque la versión en funcionamiento que tenemos del en el Crawler usa una versión de Firefox antigua dónde los permisos eran menos restringidos y con un SDK para plugins modificado que nos permite acceder desde ellos a más servicios del navegador que no estarían disponibles de otra manera. La herramienta de Crawling más concretamente, hace uso de una herramienta de análisis llamada Fourthparty que a la par de ser muy potente en muchos aspectos, está desactualizada al requerir de permisos adicionales que los navegadores más modernos no permiten.

Como nosotros queremos centrarnos en llegar al usuario, nos tenemos que adaptar al navegador que tenga este, para la primera prueba de concepto y estudiar la viabilidad del mismo vamos a trabajar con una extensión para Chrome dado que la posibilidad de conseguir acceder a los datos deseados no está asegurada.

Por lo tanto, el primer objetivo monitorizar las API's de HTML5 que nos interese.

- Mapear las diferentes técnicas de tal modo que podamos identificar las técnicas expuestas.
- Realizar el algoritmo capaz de detectar las técnicas a partir de los algoritmos.
- Reportar a nuestro servidor de backend las técnicas identificadas para el posterior análisis i generación de listas para ser difundidas entre otros usuarios.
- Realización de una UI para nuestra extensión que de información al usuario de lo que está ocurriendo.

9.2 Desarrollo

La parte crucial de este desarrollo consistía en ver si éramos capaces de interceptar las llamadas a las APIs para registrar su uso.

Por lo que el primer reto sería poder obtener una llamada y registrar su uso.

Hay que diferenciar dos casos, cuando queremos acceder a una propiedad o aun método ya que tendremos que abordarlos de manera distinta. Para poder interceptar las peticiones lo que vamos a usar es métodos estáticos de Object que nos van a permitir acceder a las propiedades de los objetos que representan las APIs i modificarlas para poder loggear su uso.

Para ello desde nuestra extensión para Chrome vamos a inyecar un archivo javascript que nos permita realizar esas configuraciones. Esta modificación se deberá realizar cada vez que se cargue una página nueva de tal manera que los scripts cargados por la misma hagan uso de las APIs modificadas por nosotros.

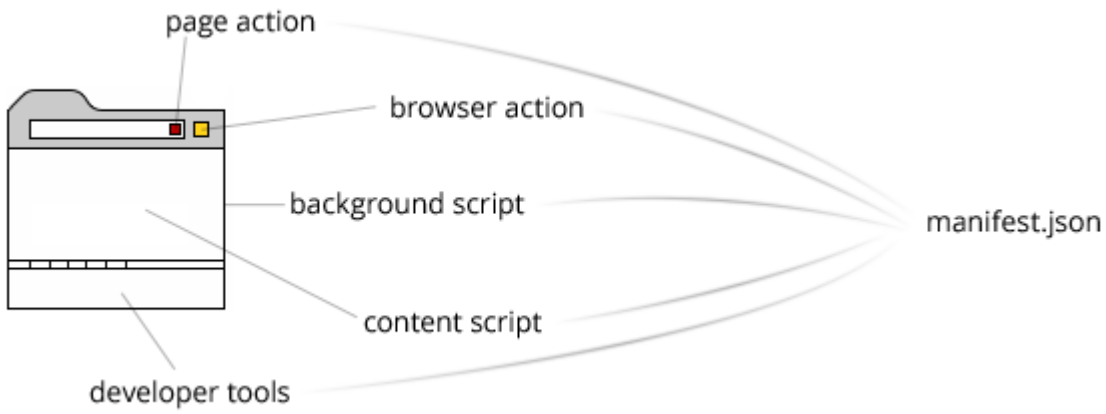


Figura 42 Partes de una extensión para chrome

Para asegurar que nuestro script se ejecuta antes que cualquier otro, usamos una de las propiedades de las extensiones que es la posibilidad de cargar un script antes de cargar cualquier otra cosa. Esta especificación se tiene que referenciar en el archivo manifest.json que es dónde se especifican los recursos, su tipo y los distintos que tiene una extensión para Chrome.

```
"content_scripts": [
  {
    "run_at": "document_start",
    "matches": ["<all_urls>"],
    "js": ["src/inject/inject.js"]
  }
]
```

Inject,js será nuestro script encargado de modificar las propiedades y métodos de nuestro interés para saber cuándo se accede a ellas, este script será como si formase parte de la web misma por lo que tendrá acceso a todo el contenido ejecutado como si del mismo sitio se tratase.

```
"permissions": [
  "tabs",
  "activeTab",
```

```
    "webRequest",
    "webRequestBlocking",
    "<all_urls>"
  ]
```

También vamos a requerir permisos especiales para las funciones que queremos realizar como se ve en el JSON superior. Vamos a requerir acceso a la información relacionada con las diferentes pestañas para poder mandar y recibir información de ellas. Saber cuál es la pestaña activa para saber a qué popup mandar la información en cada momento. Acceder a los recursos que pide un sitio web para poder bloquear su carga en caso de que sean scripts no deseados por nuestro analizador y finalmente acceso a cualquier url a la que entremos de tal modo que escanearemos todos los sitios por los que pasa el usuario ya que no tendría sentido limitarnos en ese aspecto a un rango conocido.

Inject.js va a ser el script que como hemos comentado va a ser inyectado en el sitio web. Estos scripts se conocen como content scripts en el desarrollo de extensiones para Chrome y no tienen ningún tipo de acceso que no tenga un script normal al SDK de Chrome Extensions. Lo único que nos va a permitir es mandar mensajes al script de background que en cierto modo hará de servidor en nuestra herramienta ya que también va a gestionar las comunicaciones con el popup.

En el script inyectado vamos a instrumentar las APIs que son usadas por las técnicas que analizamos de tal modo que vamos a saber cuándo son usadas. La idea es conseguir loguer sus usos sin tener que hacer uso de ninguna propiedad para desarrolladores ni nada similar.

El script inyectado es el que realiza el trabajo de reportar las APIs y sus métodos utilizados y por lo tanto es una pieza fundamental del ecosistema de la herramienta de análisis y la que más problemas ha dado a nivel de viabilidad.

Para poder detectar el uso de las APIs la estrategia que se ha seguido es sobrecargar los métodos para que lanzen un evento al ser utilizados. Este evento tendrá un identificador aleatorio por dos motivos, dificultar que un sitio bloquee su lanzamiento en función del identificador de tal modo que no se reporten los usos de las apis y por lo tanto que nuestra extensión pudiese ser bloqueada y segundo, que la ser un número aleatorio la probabilidad de colisión con otros eventos es virtualmente nula.

Para poder detectar las APIs usadas no hay una manera estandarizada para ello, la opción por la que se ha apostado en este proyecto es sobrecargar las propiedades y los métodos de las APIs de tal modo que añadiremos el lanzamiento de un evento a ellos cuando sean usados.

Para ello vamos a “instumentalizar” las diferentes propiedades accesibles de estos. La idea es automatizar el proceso lo máximo posible para pno tener que intervenir al añadir nuevas técnicas a analizar.

Por lo que vamos a trabajar con los dos tipos de datos a analizar. Propiedades (instrumentObjectProperty) y objetos (instumentObject).

Para sobrecargar una propiedad, l oque vamos a hacer es obtener su descriptor con el método estatico de Object, getOwnProperty, que pasándole como parámetro el objeto y la propiedad nos va a devolver el descriptor de dicha propiedad. Con este método podremos accdeder a los getters y setters y modificarlos de tal modo que lancen un evento cuando sean utilizados.

Para ello vamos a almacenar sus getters y setters originales en una variable temporal que nos va a permitir añadirlas dentro de las nuevas funciones que van a encargarse de realizar la misma función más el posterior reporte. Para ello definimos la nueva propiedad y hacemos una llamada a los getters y para los setters el procedimiento es muy similar devolviendo la función original después de loggear la llamada por lo que no va a existir ninguna diferencia de funcionamiento.

Para obtener la información relacionada con el script que ha hecho la petición, es necesario obtener el contexto del script. Para ello se aprovecha de una de las propiedades de la clase Error, que nos permite obtener ese contexo. Error nos permite identificar dónde se produce un error con su propiedad stack que identifica dónde se ha producido el error. Por lo que lanzamos un error para que nos devuelva el contexto del mismo de tal modo que podamos identificar el nombre del script.

El caso de la instumentación de un Objeto (prototype), el modelo seguido es exactamente el mismo, ya que al final un objeto va a ser un contender de propiedades y vamos a iterar sobre ellas para instrumentarlas.

Una vez recibimos un nuevo evento en el script inyectado debe ser reportado al script de background que va a revisar las diferentes técnicas para que tenemos mapeadas y dónde el algoritmo de detección va a ser ejecutado para ver si hay algún positivo y reaccionar a ello.

Para mandar mensajes al script de background usamos el SDK para Chrome Extensions que nos permite mandar mensajes. Hemos creado un método que usa el SDK que ya implementa la configuración para mandar los logs, estos mensajes van a poder ser leídos por todas las partes de la extensión ya que el modo de funcionamiento es el de un bus de mensajes y somos nosotros los responsables en cada parte de la extensión de decidir dónde leer esos mensajes y cómo reaccionar.

```
function sendLogMessage(log){
  chrome.runtime.sendMessage({log, origin: "symbolDetector.js", type: "ADD_LOG"}, function(response)
{
    if(response.status == "error"){
      console.log("ERROR: " + response.message);
    }
  });
```

}

Como vemos en la llamada a sendMessage, especificamos el tipo de mensaje en el campo type para posteriormente en background analizarlo.

En el script de background, vamos a procesar los diferentes mensajes recibidos ya que los tenemos de diferentes tipos para interactuar una vez se añade un nuevo log, a las acciones realizadas por el popup, la inicialización del mismo etc.

Una vez se añade un nuevo log, se busca la tabla de técnicas asociada a esa pestaña y allí rastreamos los listados en busca de si existe alguna técnica la cual contenga en nuevo símbolo logueado, la implementación de la búsqueda de símbolos en las técnicas se ha implementado de tal manera que permite búsqueda ordenada ya que sobretodo en el caso de técnicas de fingerprinting el orden es crucial para detectar un positivo dado que las acciones se deben realizar en un orden especifico. Otra opción implementada es la opción de añadir símbolos prohibidos de tal manera que, si hay un positivo de un símbolo concreto, esa técnica se descarta. Esta opción es muy útil por el hecho de que hay técnicas muy parecidas en que se pueden diferenciar en un paso peor el resto de la interacción ser común, sin esa posibilidad de bloqueo de una técnica concreta, sería inviable saber con certeza el tipo de script que lo ha producido.

Por ejemplo, para los casos de minado, CoinHive y CoinHave son dos empresas distintas que sus scripts son prácticamente idénticos, la única sutileza wue los diferencia es que CoinHave no mira el número de cores disponibles para ejecutar tantos Workers como cores disponibles, sino que directamente crea 2 Workers. La única manera de no dar un match positivo con CoinHave es determinar que no se trata de CoinHave cuando veamos que se quiere obtener el número de cores disponibles.
Una vez asociamos un símbolo a una técnica, esta se retira del array de símbolos faltantes para obtener un positivo y cuando este es de longitud cero y no hay ningún símbolo vetado, sabemos que acabamos de obtener un match y pasamos al a siguiente fase.

Una vez tenemos un positivo este se reporta a nuestro backend para su posterior análisis manual y en caso de realmente tratarse de un positivo se añade a nuestra blacklist, los positivos obtenidos en un usuario temporalmente, aunque no se hallen en la lista pasaran a formar parte de la lista local del usuario de tal forma que si vuelve a entrar en un sitio web dónde se encuentra el script, este va a ser bloqueado.

Finalmente, en background se almacenan los estados de cada pestaña de tal manera que cuando el usuario despliega el popup, este se inicializa ya que mientras no está extendido no se puede interactuar con él para informar de posibles positivos. Por lo que cada vez que lo abrimos se manda un mensaje de inicialización y es el backend que pasa un reporte.

La última parte que creo que es interesante entrar en detalle a definir es el bloqueador que es el encargado de bloquear los scripts en caso de ejecución o sustituir una librería en caso de estar corrupta con alguna técnica de las que tenemos registradas.

Esta función utiliza una propiedad muy interesante del SDK de Chrome para extensiones, que es webRequest.

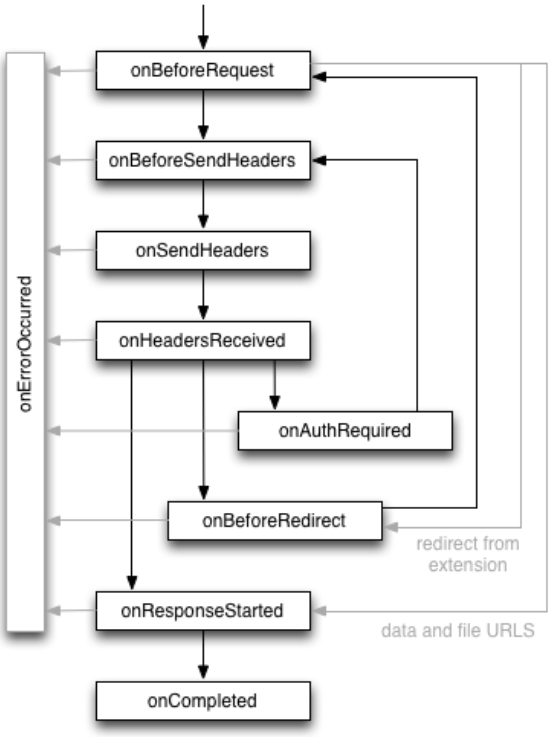


Figura 43 ciclo de vida de la carga de recursosen una extensión para chrome

Como vemos, webRequest nos permite interceptar las peticiones en un momento del ciclo de carga, en nuestro caso en particular nos va a interesar acceder al recurso antes de que sea solicitado de tal modo que así podremos bloquear su carga en caso de que esté en nuestra lista negra.

onBeforeRequest acepta 3 parámetros, una función cuyo input van a ser los detalles del recurso, una lista de urls a las que reaccionar (filtro y que en nuestro caso será nuestra lista negra) y finalmente la reacción que en nuestro caso va a ser el bloqueo.

En la función previa que podemos interactuar con los datos del recurso, podemos realizar una serie de acciones como en nuestro caso identificar si es una librería corrupta. Al terminar la ejecución de esta función debemos devolver la acción a realizar por el bloquedor, en nuestro caso tenemos dos opciones, el caso donde queremos redirigir la petición informando de la url de redirección para ese recurso en concreto (`{redirectUrl: blacklist.redirectUrl(details.url) }`), y el caso en que queremos bloquearla que simplemente informaremos que ese recurso debe cancelar su carga(`{cancel: true}`).

Mencionar también que cuando tenemos clara la reacción que vamos a tener, procedemos a lanzar la ejecución del mensaje de la acción realizada en el badger.

9.3 Funcionamiento y casos de uso

Con esta extensión por primera vez somos capaces de detectar en tiempo real técnicas invasivas en tiempo real de manera completamente eficaz. Ya que a diferencia del resto de soluciones realmente analizamos lo que está sucediendo.

Es una solución completamente transparente para el usuario que quiera darle uso, pero se ha añadido la posibilidad de añadir los análisis propios como veremos a continuación.

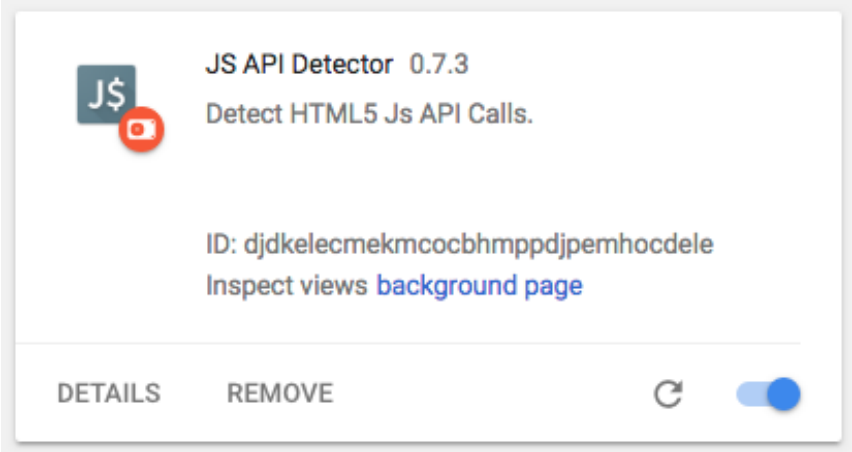


Figura 44 Panel de extensiones notificando que nuestra extensión está instalada

Una vez tengamos la extensión instalada, nos aparecerá en la barra superior del navegador. Si clicamos en el icono vamos a desplegar el popup informativo creado de tal modo que vamos a obtener información acerca del estado de la herramienta.

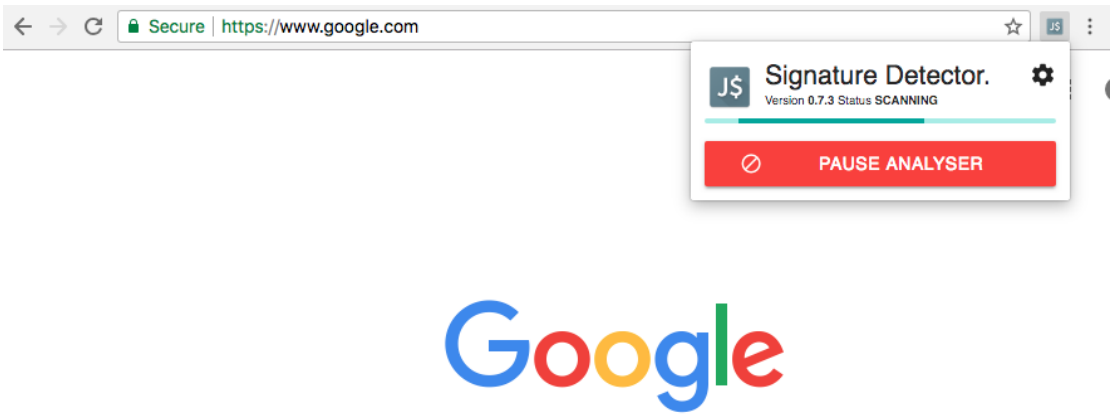
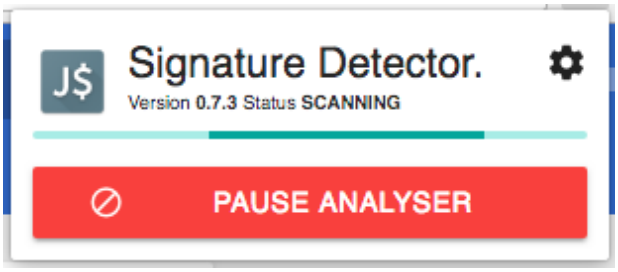


Figura 45 Popup informativo desplegado escaneando

En general tenemos tres estados diferentes para la herramienta.

- **SCANNING:** Cuando estamos escaneando en búsqueda de algún positivo.

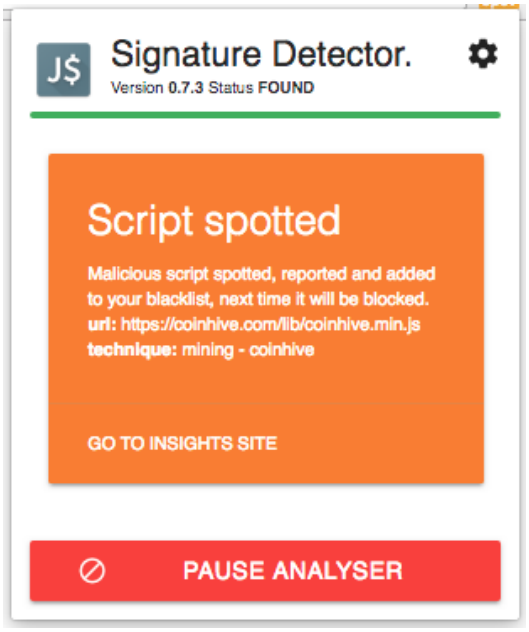


- **STTOPED:** Cuando paramos el analizador de tal modo que no va a escanear.

- **FOUND:** Para los casos cuando hay una detección o algún match con una técnica previamente detectada.

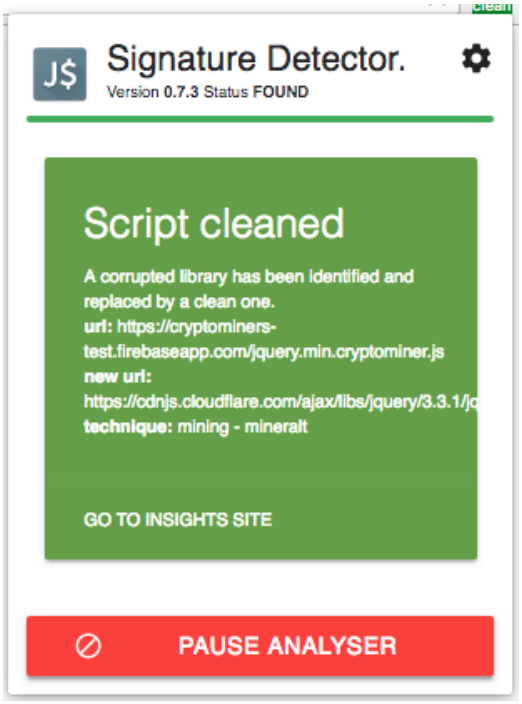
Cuando estemos navegando por sitios web y la extensión encuentre algún positivo, pasaremos al estado FOUND, que a su vez tiene 3 posibilidades distintas.

SPOTTED: Cuando detectamos una técnica, se notifica al usuario, el script malicioso no puede ser detenido dado que ya se está ejecutando y no es posible eliminar un script que ya está cargado en el sitio. Lo que hacemos es añadirlo a la lista negra para que la próxima vez que nos crucemos con ese script quede bloqueado.



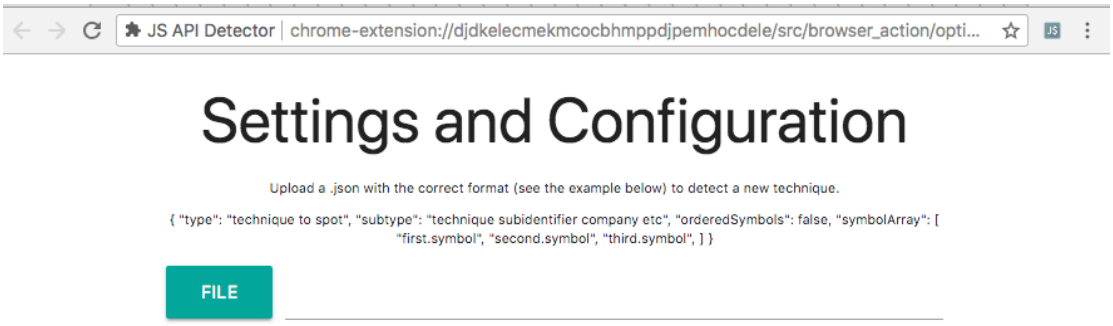
BLOCKED: Este caso surge como consecuencia de un "SPOTTED" previo o del checkeo de la lista que tenemos en el backend con las nuevas técnicas detectadas y comprobadas dado que para bloquear tenemos que detectar una técnica.

CLEANED: Este es un caso particular muy interesante. Gracias a la herramienta de Crawling y la gran cantidad de positivos distintos detectados he podido ver que algunos sitios web incluyen el script de minado dentro librerías imprescindibles como JQuery, cuando esto es detectado, la librería se sustituye por una limpia de tal modo que el funcionamiento del sitio web no se rompe.



Desde todos los positivos podemos ver un botón de "GO TO INSIGHTS SITE" que nos va a permitir ver las estadísticas globales de todos los usuarios usando la herramienta.

Finalmente, también tenemos una opción para investigadores y desarrolladores que quieran jugar con la herramienta que consistiría en añadir una técnica propia a al listado de existentes clicando en la rueda de settings.



Como podemos ver consiste en un sitio web que es ejecutado dentro de nuestra extensión, dónde podemos añadir un fichero JSON el cual debe incluir el formato previamente comentado de la estructura de una técnica. Una vez hecho esto la extensión añadirá la información a la base de datos para ser posteriormente escaneada.

10. Metodologías y desarrollo

Aunque este proyecto lo haya realizado sólo, una idea era aplicar los conocimientos de metodologías ágiles en la realización del mismo. Par ello, se ha trabajado con sistemas de versionado git en todas las diferentes partes del proyecto. En concreto se ha usado el sistema de versionado GitLab ya que nos permite tener repositorios privados y utilizar su integración continua de manera gratuita como comentaremos en el siguiente apartado.

10.1 Realización de entornos de producción

Para las herramientas se han utilizado entornos de integración, concretamente la integración continua que nos ofrece GitLab.

GitLab nos ofrece la posibilidad de automatizar una serie de tarea sal hacer un push a una rama mediante un fichero de configuración YAML. El siguiente es el script de integración para el script de fingerprinting a modo de ejemplo.

```
image: node:boron

stages:
  - install
  - build
  - deploy

before_script:
  stage: install
    - npm install -g firebase-tools
    - npm install -g gulp
    - npm install -g gulp-concat
    - npm install -g gulp-rename
    - npm install -g gulp-uglifyes

global_scope: #set variables for global usage
  stage: build
environment: Production
  only:
    - production
  script:
    - npm link gulp
    - npm link gulp-concat
    - npm link gulp-rename
    - npm link gulp-uglifyes
    - gulp
    - cp -R src/js/ build/js/
    - cp src/test.html build/


deploy_to_production:
  stage: deploy
    - firebase use --token $FIREBASE_DEPLOY_KEY
```

```
    - firebase deploy -m "Pipeline $CI_PIPELINE_ID, build $CI_BUILD_ID"
--non-interactive --token $FIREBASE_DEPLOY_KEY
```

Como vemos nos permite generar unos tages que van a ser las diferentes fases que se van a producer, en este caso vemos que tenemos la fase de instalación de los modulos que van a ser instalados en el Docker que nos da GitLab. Como vemos son paquetes npm ya que las tareas las vamos a realizar en node.

Luego pasamos a la fase de build donde vemos que se ejecuta gulp que va a realizar una serie de tareas que nosotros hemos programado como juntar todos los archivos JavaScript en uno que va a ser la librería (se han desarrollado por separado para facilitar el testing de los diferents fingerpints en este caso) y posteriormente se van a renombrar le fichero y se va a ofuscar para que no sea fácilmente detectalbe (necesario para realizar tests reales) ni que nuestro código pueda ser fácilmente analizado por terceros.

Finalmente tenemos la fase deploy que en nuestro caso se va a encargar de subir los ficheros a un hosting desde el cual podrán ser accedidos y ser usados como una librería.

Como vemos se usa la herramienta de Firebase para NodeJs y para no tener que usar sus claves en código y que estén disponibles para todo el mundo con acceso a al repositorio, se han creado unas variables internas de GitLab en integración continua que son accedidas para obtener los valores sin problemas.

Finalmente, si las operaciones realizadas tienen éxito, se finaliza el test con un éxito.

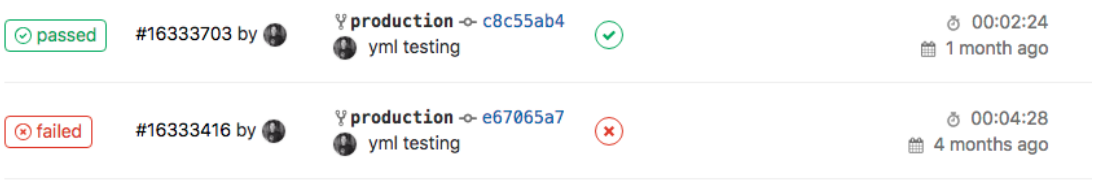


Figura 46 ejemplo de caso de test con éxito y test con errores

10.2 Flujo de trabajo (GitFlow)

A pesar de en este proyecto no estar trabajando con un equipo de desarrollo se han intentado seguir de todos modos las prácticas de desarrollo de un proyecto de tal modo que nos van a permitir una integración más sencilla en un futuro si terceros se incorporan al proyecto y también favorece a que el ciclo de trabajo quede más limpio.

GitFlow nos permite dimensionar un proyecto de manera correcta para llegar a deadlines como es el caso.

Estos proyectos al no tener tareas simultaneas puesto que yo era el único colaborador, no san creado ramas de desarrollo y las funcionalidades se han ido realizando sobre la rama máster para luego al disponer de una realase, usarla en la rama production dónde va a producirse toda la integración continua para paquetear y hacer el deploy de la herramienta concreta como se comenta en el apartado anterior.

10.3 Tecnologías usadas

En este proyecto se ha trabajado con diversas tecnologías.

El crawler se ha trabajado con el lenguaje bash para el proceso de automatización, en Python para modificar el crawler y la generación de los scripts de análisis entre otros y en JavaScript para modificar la extensión que nos permite recabar los símbolos JavaScript ejecutados y añadir los que sean de nuestro interés.

Para el Scrapping de sitios web se ha trabajado en Python y se ha generado una base de datos en MongoDB con los datos obtenidos.

Los datos obtenidos del Crawling se almacenan en una base de datos PostgreSQL y una vez analizados los positivos obtenidos son añadidos a nuestra base de datos mongo.

Para el backend se ha trabajado con Firebase y Firebase Functions que nos permiten tener el backend en la nube. Para poder trabajar con Firebase Functions el código del servidor se ha realizado con NodeJs.

El script de fingerprinting y la interfaz web de resultados ha sido realizada con tecnologías HTML5 y en concreto en esta segunda se ha realizado un uso extensivo de librerías de diseño como es la librería D3 que no permite generar mediante JavaScript cualquier tipo de grafico Vectorial.

Finalmente, para la extensión para Chrome se ha trabajado con tecnologías HTML5, conceptos y Apis propias de desarrollo de extensiones para poder realizar las comunicaciones necesarias.

Todas las comunicaciones mediante los backends y las aplicaciones se han realizado mediante la generación de APIs RESTful que posteriormente pueden ser útiles para futuros desarrollos gracias a su polivalencia.

La integración continua de los repositorios de las herramientas se ha realizado en NodeJS y los múltiples módulos usados.

11. Conclusiones

Conversando con Brendan Eich durante la conferencia anual del Data Transparency Lab en la universidad de Columbia(NY), considerado el creador de JavaScript, ex-CEO de Mozilla Corporation y actualmente CEO el navegador Brave (brave.com) centrado en dar privacidad a los usuarios, pude constatar que la privacidad, siendo muy importante y esencial, ya no tiene únicamente valor como tal, sino que puede ser la medida más importante para aumentar el rendimiento de los sitios web, ya que las medidas de tracking actual consumen grandes cantidades de recursos tanto de red como del dispositivo cliente y obviamente las técnicas de minado consumiendo ancho de banda y poder computacional son aún peores en ese sentido.

Su bloqueo puede aumentar de manera sustancial el rendimiento de la web, lo que es una perspectiva muy interesante que no se ha considerado habitualmente.

La conclusión que puedo sacar de este proyecto, es que la industria dedicada a la obtención de datos personales crece a un ritmo difícil de seguir por reguladores, periodistas especializados, políticos y sobre todo el usuario medio.

Herramientas como las desarrolladas en este proyecto son imprescindibles para cuantificar las técnicas utilizadas, o incluso revelar la utilización de técnicas desconocidas hasta el momento.

Con la herramienta de Crawling actual no solo se pueden estudiar las técnicas de rastreo mediante fingerprint o identificar el minado de criptomonedas, sino que es posible el análisis de prácticamente cualquier técnica, por su capacidad de monitorizar, extraer y guardar en la base de datos todo el flujo de transferencia de datos entre el cliente y el servidor.

El futuro de la herramienta de crawling es por lo tanto el análisis continuo y periódico del máximo número de sitios web que nos permita el hardware del servidor, y ver cómo evoluciona el uso de las diferentes técnicas. Simultáneamente, la investigación en los nuevos métodos usados por la industria, permitirá actualizar y obtener una herramienta cada vez más completa, que sirva para auditar la privacidad de un sitio web, con el objetivo final de proteger y beneficiar al usuario al mismo tiempo que se le ofrece un conocimiento más detallado de lo que está sucediendo.

A nivel personal, los conocimientos que he adquirido en cuanto a privacidad de datos durante mi estancia en el Data Transparency Lab de Telefónica, no únicamente en el campo de fingerprinting, cookies, mining o privacidad en la web en general, sino también en la misma red o en aplicaciones móviles gracias a proyectos desarrollados por otros departamentos, han sido de un valor incalculable.

11.1 Desarrollo futuro y posibles mejoras

Tras la conferencia del Data Transparency Lab 2017 (BCN), y ver el panel de Brave. Vi que sus proyectos iban muy relacionados con este proyecto en concreto.

Figura 47 Brave en desarrollo en detección de fingerprinting y minado

Hablando con ellos, me comentaron que su modo de uso era mediante listas por lo que les pareció muy interesante el enfoque que le estábamos dando a la probelmática ya que la solución era mucho más potente.

Tras las conversaciones que mantuvimos con la gente de Brave, Brendan Eich y Ben Livshits (Chief Scientist), uno de los pasos hacia dónde nos gustaría evolucionar la extensión pensada para que los usuarios finales pueden bloquear las técnicas estudiadas es hacia un SDK utilizable por terceros.

Aunque nos hemos centrado en el desarrollo para Chrome, y que por lo tanto ya cubriríamos todo el abanico de navegadores basados en Chromium como son Chome, Brave, Opera o Yandex, sería muy interesante tener la misma herramienta para navegadores basados en Firefox como Cliqz de tal modo que pudiésemos automatizar un Crawling con cada navegador de tal manera que nos permitiera realizar un estudio fiable y confiable de la protección que ofrece cada uno.

En la misma línea una herramienta de Crawling más eficiente completamente diseñada por nosotros que nos permitiera realizar el Crawl con el navegador que quisiéramos.

Otro objetivo a corto plazo, es realizar el test de minado mensualmente y ver la evolución del uso de esta tecnología tan incipiente, ver si se confirma como una manera de monetización complementaría a los anuncios, si ha quedado en una oleada o si se mantiene en un porcentaje reducido de los sitios.

Respecto a la herramienta para clientes, sería interesante calcular el tiempo que el usuario pasa en el sitio dónde hay una detección para estimar cuánto dinero ha generado y ver la gravedad del problema.

12. Bibliografía

[1] K. M. a. H. Shacham, «Pixel Perfect: Fingerprinting Canvas in HTML5,» University of California, 2014.

[2] J. Schmidt, «heise.de/,» [En línea]. Available: https://www.heise.de/ct/ausgabe/2018-9-Krypto-Miner-Ihr-PC-rechnet-fremd-4013390.html.

[3] A. Rosic, «blockgeeks.com,» [En línea]. Available: https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/.

[4] PostgreSQL, «formatos PostgreSQL,» 2016. [En línea]. Available: https://www.postgresql.org/docs/8.2/static/functions-formatting.html.

[5] P. G. Pérez, «elladodelmal.com,» [En línea]. Available: http://www.elladodelmal.com/2018/01/coffeeminer-te-tomas-tu-cafe-te.html.

[6] B. Packets, «badpackets.net,» 2017. [En línea]. Available: https://badpackets.net/how-to-find-cryptojacking-malware/.

[7] L. O'Donnell, «threatpost.com,» [En línea]. Available: https://threatpost.com/ad-network-circumvents-ad-blocking-tools-to-run-in-browser-cryptojacker-scripts/130161/.

[8] S. E. a. A. Narayanan, «Online Tracking: A 1-million-site Measurement and Analysis,» Princeton University, 2016.

[9] Mozilla, «AudioContext API,» 2016. [En línea]. Available: https://developer.mozilla.org/en/docs/Web/API/AudioContext.

[10] K. Leuven, «Cookieless Monster,» Leuven, Belgium, 2013.

[11] D. Fifield, «Fingerprinting web users through font metrics,» University of California, 2015.

[12] S. Englehardt, «OpenWPM: An automated platform for web privacy measurement,» Princeton University, 2015.

[13] P. Eckersley, «How Unique Is Your Web Browser?,» 2010.

[14] E. Chong, «fortinet.com,» [En línea]. Available: https://blog.fortinet.com/2018/02/07/the-growing-trend-of-coin-miner-javascript-infection.

[15] J. Cazala. [En línea]. Available: https://github.com/cazala/coin-hive-stratum.

[16] A. Barth, «ITEF HTTP State Management Mechanism,» U.C. Berkeley, 2011.

[17] Alexa, «Alexa Ranks,» 2017. [En línea]. Available: http://www.alexa.com/.

[18] G. Acar, «The Web Never Forgets:,» KU Leuven, 2014.

[19] [En línea]. Available: https://github.com/cryptonoter/CryptoNoter/tree/master/web.