MÁSTER EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DECENTRALIZED APPLICATIONS' SAFETY AND SECURITY

Autor: Miguel Oleo Blanco

Director: Yue Duan

Co-Director: Sajad Meisami

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Decentralized applications' safety and security

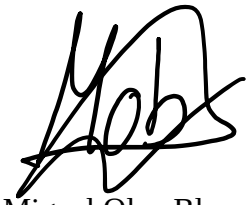
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/2023 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Miguel Oleo Blanco Fecha: 11/07/2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Yue Duan Fecha: 11/07/2023

EL CO-DIRECTOR DEL PROYECTO

Fdo.: Sajad Meisami Fecha: 11/07/2023

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DECENTRALIZED APPLICATIONS' SAFETY AND SECURITY

Autor: Miguel Oleo Blanco

Director: Yue Duan

Co-Director: Sajad Meisami

Madrid

Agradecimientos

Quiero recalcar en esta sección la ayuda de mi compañero de proyecto Hugo Dabadie, compañero del IIT. Con el hemos conseguido llevar a delante este proyecto o sobrepasar todos los retos que nos han ido surgiendo a lo largo del desarrollo de este. Le deseo mucha suerte con la finalización de su proyecto y con su futura carrera profesional.

DECENTRALIZED APPLICATIONS' SAFETY AND SECURITY

Autor: Oleo Blanco, Miguel
Director: Duan, Yue.
Co-Director: Meisami, Sajad

RESUMEN DEL PROYECTO

En este trabajo se investigará sobre la seguridad de las aplicaciones distribuidas o DApps. Para ello se realizará un estudio previo sobre los distintos mecanismos de seguridad que la cartera digital Metamask proporciona a las aplicaciones. Con los resultados obtenidos, se creará una herramienta de análisis estático del código para determinar la seguridad de estas.

Palabras clave: Blockchain, DApps, Ciberseguridad, Criptomonedas, Testing.

1. Introducción

Desde la aparición de la tecnología de la cadena de bloques, muchos problemas de seguridad han surgido. Estos problemas normalmente vienen vinculado a estafas y fallos de seguridad vinculados a las aplicaciones descentralizadas que trabajan sobre el blockchain. Debido a esto, hay una necesidad por parte del usuario de fiarse de estas aplicaciones que, finalmente, pueden resultar maliciosas. Por ello, con este trabajo, se busca elaborar una herramienta que, analizando el código fuente de dichas aplicaciones, determine si es seguro de utilizar. Para ello, es necesaria una investigación previa sobre la implementación de los mecanismos de seguridad y su funcionamiento.

2. Definición del proyecto

El principal objetivo de este proyecto es la creación de una herramienta que, analizando el código fuente de una aplicación distribuida especificada, determine su seguridad. Para ello, como se ha citado en la sección de la introducción, hay una fase previa de investigación. Es esta fase primero se estudió el lenguaje de programación más usado entre las 100 DApps más usadas.

Una vez estudiado el lenguaje de programación, se estudian los diferentes métodos/paquetes para implementar el blockchain con las aplicaciones. En esta fase se concluye que, independientemente del paquete, es la propia wallet la que gestiona los permisos y verificaciones. Debido a esto, se procederá a estudiar en profundidad Metamask, ya que es la principal wallet. Estas wallets incorporan una serie de APIs propias y heredadas de Ethereum (o de otras criptomonedas). Para comprender en detalle el funcionamiento de estas APIs, se ha creado una tabla con los 70 diferentes métodos, los parámetros obligatorios y opcionales, si es necesaria confirmación y el resultado de este. En esta fase, se concluye que la documentación, al no haber una organización

principal detrás de definición de estos métodos, las implementaciones pueden tener pequeñas diferencias con las recomendaciones de Ethereum (EIP). Una vez detectadas las APIs necesarias de confirmación, se realiza un estudio en profundidad de las mismas. Para ello, se analiza el código fuente de estas y se compara con el proporcionado por la documentación de Metamask.

Una vez entendido el código, se procede a buscar la implementación de tres métodos de firma en el listado de las 100 DApps. Estos tres métodos son *eth_sign, eth_signtypeddata* y *personal_sign*, haciendo hincapié en el segundo. Para el estudio del método *eth_signtypeddata* se ha creado una aplicación para test en la que se han probado distintos contenidos para determinar la seguridad de este método. Con estas pruebas se ha logrado determinar la estructura mínima necesaria para que la firma resultante al ejecutar este método sea exitosa. También, se ha logrado terminar que, debido a la flexibilidad que ofrece este método, no se establecen parámetros obligatorios ni se comprueban dichos parámetros.

Más adelante en el proyecto, para conseguir una muestra más representativa del "mercado", se ha ampliado esta base de datos de DApps a 771 aplicaciones. Para ello, se han empleado una serie de *scripts* y herramientas para la obtención de aplicaciones de forma masiva. Una vez se dispone de esta base de datos, se procede a estudiar en profundidad los distintos métodos citados anteriormente. Para ello, se va anotando los resultados en la misma base de datos para sacar conclusiones.

Durante el estudio de estos métodos, se cita el proceso empleado, así como su resultado y las razones por las que se consideran seguros o vulnerables (citados en los resultados). Por último, se crea una herramienta capaz de analizar (en tiempo real) todas las aplicaciones usadas para la creación de la base de datos. Los resultados se muestran al usuario con una interfaz gráfica y se explica el motivo, en caso de haberlo, de porqué la aplicación investigada es vulnerable o no. En el apartado de resultados se comenta todo lo obtenido durante el proyecto.

3. Descripción del modelo/sistema/herramienta

La metodología empleada para el correcto desarrollo del proyecto se basa en la metodología *Agile*. En esta metodología se crean muchos *Sprints* (pequeñas iteraciones del trabajo), consiguiendo reiterar el trabajo del trabajo e incorporar e incrementar el contenido de este. Esta metodología es la óptima para este tipo de trabajos ya que, como se trata de un trabajo de investigación, es difícil planificar todo desde un principio, ya que hay muchas incógnitas que surgen a lo largo del proyecto. Para aplicar esta metodología en el proyecto, se ha optado por planificar al menos una reunión por semana (dos preferiblemente). En dichas reuniones, se comentan los resultados obtenidos durante la semana y se discute y fijan los objetivos a obtener para la siguiente reunión.

Como se ha citado previamente, se trata de un trabajo de investigación. Esto conlleva que la mayor parte del proyecto se centra en conocer la herramienta que se emplea y el objetivo. Debido a esto, no es hasta el final del proyecto donde se obtiene un resultado

mesurable. Otro de los objetivos del proyecto, es contribuir a la comunidad científica con la publicación de un *paper* con los resultados obtenidos. Este último objetivo, por motivos de tiempos, se ejecutará pasada la defensa de este trabajo.

En cuanto a las herramientas que se utilizan a lo largo del desarrollo del trabajo, hay una gran variedad. Cada herramienta proporciona una utilidad distinta (el cual se citará a continuación) al desarrollo del proyecto:

- Metamask [1]: Esta es la principal herramienta del proyecto. Esta es una cartera digital de criptomonedas que se conecta a las distintas aplicaciones descentralizadas y proporciona toda la seguridad, lógica y funciones a estas. De esta herramienta, se investigan las funciones de seguridad y aquellos métodos que requieren confirmación del usuario. Estos últimos son los más importantes, ya que se tratan de métodos con datos sensibles y por donde se introducen la gran mayoría de ataques.
- DappRadar [2]: Esta es una página web que nos proporciona información sobre las DApps con más tráfico (las más usadas) del internet. Esta herramienta nos proporciona la información necesaria para investigar el lenguaje de programación más usado de las aplicaciones. Este listado se ha utilizado para llevar a cabo la investigación sobre la implementación de los distintos métodos estudiados.
- Google Drive: Esta herramienta se ha incorporado al proyecto para compartir la información que se va generando. Nos proporciona una plataforma para crear documentos y compartir con el resto de los miembros del proyecto. Principalmente se ha hecho uso de las hojas de cálculo tipo Excel para recoger toda la información sobre los distintos métodos de Metamask estudiados. Esto nos permite listar toda la información de forma organizada.
- Github [3]: Esta plataforma aporta un repositorio virtual en internet donde tener disponible todo el material empleado. La principal diferencia con Google Drive, es que esta plataforma está enfocada al código fuente. También proporciona funciones específicas para equipos de desarrollo.
- Node.js [4]: Este es un entorno de desarrollo y ejecución para JavaScript. La gran mayoría de aplicaciones descentralizadas, están desarrolladas en JavaScript o TypeScript (lenguaje basado en JavaScript). Esta herramienta se ha empleado para el desarrollo de una aplicación propia para test que se realizan a lo largo del proyecto.
- CodeQL [5]: Esta herramienta permite, añadiendo una serie de bases de datos de repositorios de código, realizar unas series de peticiones (queries) sobre dichos repositorios. Esta herramienta es muy potente ya que estas peticiones se programan en un archivo .ql y eso permite muchísima flexibilidad de cara a hacer la petición necesaria. De cara a este proyecto, se empleará esta herramienta para recabar toda la información de seguridad de las bases de datos de aplicaciones descentralizadas. Con esta información, se podrá crear una conclusión sobre la seguridad de las aplicaciones que, posteriormente se publicará en un *paper*.

4. Resultados

- El primer resultado obtenido es la documentación de las APIs de Metamask, Ether.js y Web3.js. Este resultado es producto de la investigación llevada a cabo para el análisis de los métodos relacionados con la seguridad. Este resultado obtenido es muy importante, ya que la documentación disponible de estos métodos es casi escasa y puede ayudar al resto de la comunidad.
- Para analizar alguno de los métodos estudiados en este proyecto, ha sido necesario crear una aplicación descentralizada. Con esta aplicación, se han realizado distintas pruebas sobre el método *eth_signTypedData* y el contenido de las firmas. Esta aplicación dispone de conexión con Metamask, selección de la versión que se desea implementar del método y una caja de texto donde introducir el contenido de la firma. Gracias al *popup* de Metamask y el display de la aplicación sobre si el resultado de la firma es exitoso o no, se han conseguido sacar las conclusiones sobre la seguridad de este método.
- Una vez comprobados los distintos métodos que están en el foco del estudio de las vulnerabilidades, se ha documentado todos los mecanismos que emplean y que *features* hacen que sean vulnerables. Este es uno de los puntos clave para el desarrollo del proyecto.
- El resultado principal de este trabajo es una aplicación que analiza la aplicación descentralizada seleccionada y, aplicando todo el conocimiento aprendido durante la fase de investigación, determina su seguridad. Para ello, se informa al usuario de la seguridad de la aplicación, asi como el motivo del resultado, en una interfaz gráfica muy simple.
- Por último y gracias a la aplicación desarrollada, se han obtenido una serie de estadísticas sobre la seguridad de las aplicaciones descentralizadas.

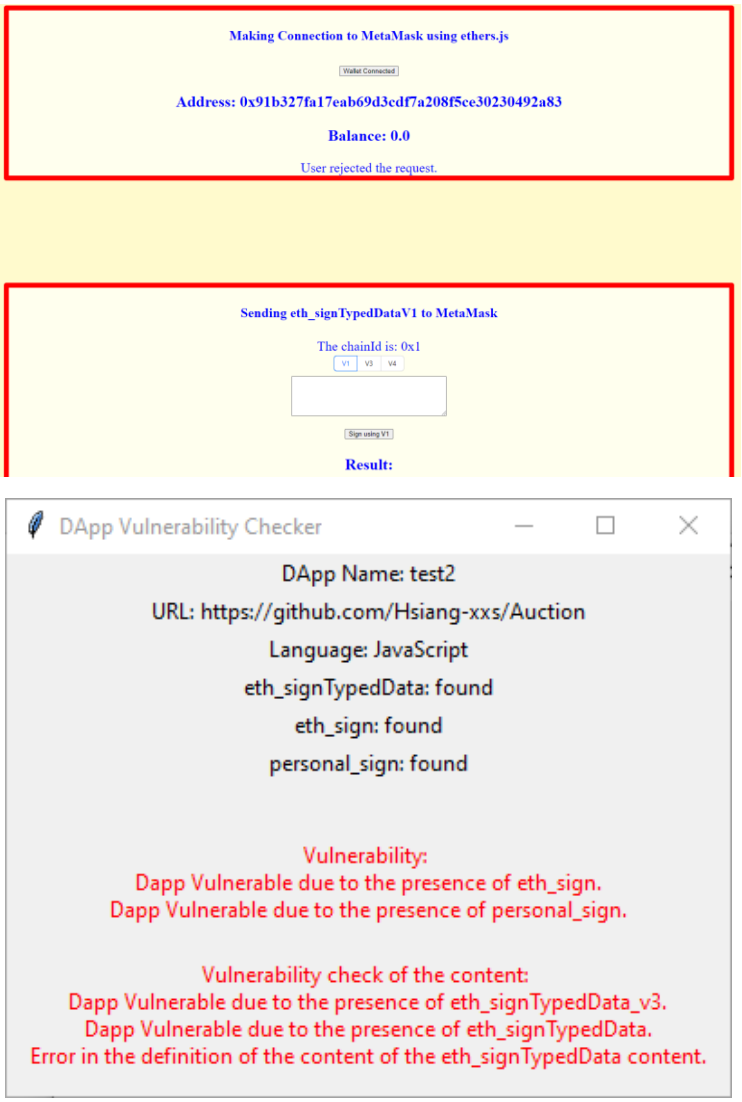


Ilustración 1: DApp desarrollada para las pruebas y herramienta de análisis de seguridad desarrollada

5. Conclusiones

La principal conclusión es que, debido a la implementación de los métodos deprecados o a versiones antiguas, un 20% de las aplicaciones analizadas resultan ser potencialmente vulnerables. Esto se debe principalmente a los métodos *eth_sign* el cual fue el primer método implementado y que tiene una serie de problemas de seguridad. Entre estos problemas se encuentra que, la firma mostrada al usuario no es legible, por lo que es fácilmente manipulable sin que el usuario final se percate de ello. También es problemática ya que emplea la misma curva elíptica para la creación de las claves, para distintos dominios de seguridad. Debido a esto, la probabilidad de que la clave privada sea descubierta crece exponencialmente. Por último, este método está deprecado, por lo que no tiene mantenimiento y esto siempre conlleva muchos ataques que aparecen con el paso del tiempo.

Por otro lado, *personal_sign* arregla el problema de la legibilidad de la firma, pero es un método muy complejo de implementar. También sigue teniendo el problema del dominio de las firmas, por lo que está expuesto a un ataque de tipo *replay*.

Por último, *eth_sigTypedData* es el método más reciente y está compuesto de las versiones 1, 3 y 4. Solamente se considera segura la última versión, ya que implementa un separador de dominio. También implementa mucha flexibilidad de cara al desarrollador, donde el contenido de este método es muy programable. Esta característica conlleva que no se comprueba este contenido y, se puede crear de forma vulnerable.

Debido a lo citado, la aplicación, cuando analiza una Dapp, considera vulnerable cualquier aplicación con presencia de los métodos *eth_sign* y *personal_sign*. Cuando detecta la presencia de *eth_signTypedData*, si se trata de las versiones 1 o 3, se reporta como vulnerable y cuando se trata de la versión 4, se realiza un análisis en profundidad del contenido de la firma para determinar su seguridad.

Con todo lo citado y con el resultado de 17% de las aplicaciones estudiadas siendo vulnerables, un 3% extra siendo potencialmente inseguras, se concluye que se trata de un "mercado" que requiere mayor integración de mecanismos y sistemas de seguridad. Esto se debe principalmente a que, estamos ante un sector que mueve miles de millones de euros y que tiene expectativas de ser el soporte base de los nuevos sistemas bancarios, sistemas de trazabilidad, etc. Por ello, y gracias a estos resultados, se llega a la conclusión de que es exitosa la fase de investigación y las hipótesis iniciales de que estas aplicaciones son inseguras comparadas con otros segmentos del mercado.

6. Referencias

[1] The crypto wallet for Defi, Web3 Dapps and NFTs | MetaMask. (s. f.). https://metamask.io/

[2] DappRadar - The World's Dapp Store | Blockchain Dapps Ranked. (s. f.). DappRadar. https://dappradar.com/

[3] GitHub: Let's build from here. (s. f.). GitHub. https://github.com/

[4] Node.js. (s. f.). Node.js. https://nodejs.org/es

[5] CodeQL. (n.d.). https://codeql.github.com/

DECENTRALIZED APPLICATIONS' SAFETY AND SECURITY

Author: Oleo Blanco, Miguel.
Supervisor: Duan, Yue.
Co-Supervisor: Meisami, Sajad

ABSTRACT

This is a research project regarding the security of the distributed apps of the blockchain, also known as DApps. The first phase objective is to research the different security methods provided by the blockchain Metamask to the DApps. With the results obtained, a static análisis tool will be created to run over DApps source codes and determine their security.

Keywords: Blockchain, DApps, Cybersecurity, Cryptocurrencies, Testing.

1. Introduction

Since the appeal of the Blockchain technology, many security problems have emerged. These issues are usually linked to phishing attacks, scams and implementation faults with the decentralized apps that works on top of the blockchain. Due to this, there is an increasing need for the users to check for the security of those apps, which can be malicious. This Project seeks to create a tool to check the source code of the apps in order to determine if they are safe to use by the final user. To achieve this, an investigation and research should be made over the topic of the security of the dapps and the blockchain wallets and how those methods are implemented on real world decentralized applications. Lastly, a tool will be created to achieve it.

2. Project Definition

The main goal of the project is creating a tool that determines whether a decentralized application is safe to use or not. This will be achieved by analyzing the source code of the application. To achieve this, all the work mentioned on the Introduction should be made, this includes all the research before the implementation of the tool. The research phase starts with the investigation of the most common programming languages used by the DApps by analyzing the ranking of the top 100 used DApps.

Once that the most common programming languages is defined, different methods and packages used to implement the blockchain into the applications, are researched. In this phase, it is concluded that, independently of the package used, the wallet is what handles all the logic, methods, permissions and verifications of the application. Due to this, the next step is deeply studying in detail Metamask, due to the fact that it is the most used blockchain wallet. Metamask implements 70 different APIs, a lot these methods are studied in detail and documented into a Google Spreadsheet with the required and optional parameters, whether the user needs to confirm or not, and the return of the

method. This phase concludes with the documentation that was obtained during the process. There are no organizations behind the blockchain, so the documentation, in most cases differs from one site to another. For that reason, the documentation obtained during the research, is key for the success of the project. In the documentation all of this is stated with the differences that were found in between the experiments we carried out and the documentation form the Ethereum Implementation Proposals (EIP). Once all the APIs that need user confirmation, are documented, a deeper study is made into those. This includes mainly studying the source code and implementation provided by Metamask and comparing it with the documentation obtained and provided.

Once the coded is understood, the next step is to look for the implementation of three of the main APIs into the top 100 Dapps. These three methods are *eth_sign*, *eth_signtypeddata* and *eth_personal_sign*. In this project, the second method is the one that we focus the research, by implementing a test DApp that allows us to try out how the wallet checks all the parameters and data. Thanks to these tests, it is determined that this method is very powerful, modern, and especially flexible. Most of the parameters are optional, which makes the method very flexible, but also very hard to make it secure, as the wallet cannot check all the data, only the definition of the structure and types.

Later in the project, to obtain a more representative sample of the "market," this DApps database has been expanded to 771 applications. To achieve this, a series of scripts and tools have been used to obtain applications on a massive scale. Once this database is available, the different methods mentioned earlier are thoroughly studied. The results are then recorded in the same database to draw conclusions.

During the study of these methods, the employed process is mentioned, along with its outcome and the reasons why they are considered secure or vulnerable (as stated in the results). Finally, a tool is created capable of analyzing (in real-time) all the applications used for the creation of the database. The results are presented to the user through a graphical interface, and the reason for whether the investigated application is vulnerable or not is explained, if applicable. The results section discusses everything obtained throughout the project.

3. Description of the model/system/tools

The methodology behind this project is key to the correct development of itself and it is based on the Agile methodology. With this methodology there are many Sprints (small iterations of the work). With this methodology, the work is iterative, and we can implement new features on top of the previous sprints. This way of working is optimum for this type of research projects, due to the fact that we keep incorporating things that we find during the research. To implement this, several meetings will be held each week to discuss the results obtained during that week and the new objectives will be set.

As mentioned before, this is a research project. That means that most of the duration of itself is occupied by the research part. Another important part is understanding the tools that will be used to research and to obtain the results. The main objective of this project is to create a paper in which all the results obtained will be mentioned and explained for the community. Due to timming, this last objective might be reached after the defense of the project.

There are several tools that will be used during the project to achieve the objectives. Each tool will provide a unique utility (that will be mentioned) to the correct development of the project.

- Metamask [1]: This is the main tool used in the project. This is a digital wallet of the blockchain that provides all the interface to the users. This wallet also provides logic, utility, and security to the decentralized applications. There will be investigated all the security features and methods that this wallet provides, especially the ones that required user confirmation. These methods are the most sensitive ones because they work with private data, and this is the entry points of many attacks.
- DappRadar [2]: This is a website that provides information about DApss with the most traffic (the most used) of the internet. This tool provides the information needed to research about the most used coding language for DApss. Also, we have created a database with the top 200 used dapps to research about the main methods that we cover in this project.
- Google Drive: This tool is used to share all the information that is being generated during the project. This tool provides a platform to create and share documents with the other team members. Mainly, we have used the spreadsheets to create and structure the database of the DApps and APIs. This is very useful to structure all the data.
- Github [3]: This platform provides a virtual repository on the internet in which we can store, share, and collaborate in coding project. The main difference with Google Drive is that this platform is focused on source codes. It also provides some specific features for coders.
- Node.js [4]: This is a development and execution environment for JavaScript. Most of the apps are coded using JavaScript or TypeScript (it is a modified version of JavaScript). This tool has been used to create a test DApps to test different methods and carry on the security tests to obtained all the data beforehand.
- CodeQL [5]: This tool enables, using several code databases from GitHub repositories to code and run queries on that database. This is a very powerful tool, as it the queries can be coded with something like Python into a .ql file. This means that, the flexibility of the queries is very high. In the context of this project, this tool will be used to generate all the information regarding the security of the DApps. All this information will be later presented into a scientific paper to the community.

4. Results

- The first result obtained is the documentation of the Metamask, Ether.js, and Web3.js APIs. This result is a product of the research conducted for the analysis of security-related methods. This obtained result is crucial as the available documentation for these methods is scarce and can assist the rest of the community.
- To analyze some of the methods studied in this project, it was necessary to create a decentralized application. With this application, different tests have been performed on the eth_signTypedData method and the content of the signatures. This application has a connection to Metamask, the option to select the desired implementation version of the method, and a text box to enter the signature content. Through the Metamask popup and the application's display indicating whether the signature result is successful or not, conclusions about the security of this method have been drawn.
- Once the different methods that are the focus of vulnerability study have been verified, all the mechanisms they employ and the features that make them vulnerable have been documented. This is one of the key points for the project's development.
- The main outcome of this work is an application that analyzes the selected decentralized application and, applying all the knowledge gained during the research phase, determines its security. The user is informed about the application's security and the reason for the result through a very simple graphical interface.
- Finally, thanks to the developed application, a series of statistics regarding the security of decentralized applications have been obtained.

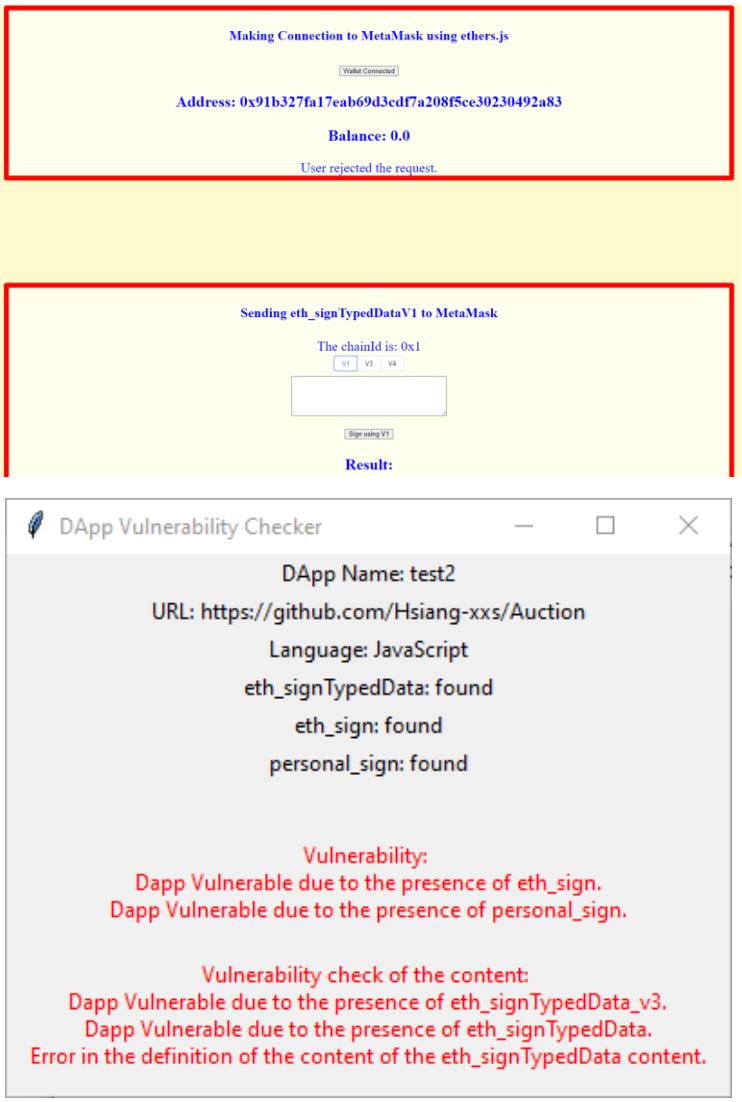


Ilustración 2: DApp developed for tests and tool develop to determine vulnerabilities

7. Conclutions

The main conclusion is that, due to the implementation of deprecated methods or outdated versions, 20% of the analyzed applications are potentially vulnerable. This is primarily due to the eth_sign method, which was the first implemented method and has a series of security issues. Among these problems is that the signature displayed to the user is not readable, making it easily manipulable without the user's knowledge. It is also problematic because it uses the same elliptic curve for key generation across different security domains. As a result, the probability of the private key being discovered increases exponentially. Finally, this method is deprecated, meaning it is no longer actively maintained, which leaves it susceptible to various attacks over time.

On the other hand, personal_sign addresses the issue of signature readability but is a complex method to implement. It still suffers from the signature domain problem, making it vulnerable to replay attacks.

Lastly, eth_sigTypedData is the most recent method and is composed of versions 1, 3, and 4. Only the latest version is considered secure as it implements a domain separator. It also provides a lot of developer flexibility, where the content of this method is highly programmable. This flexibility means that the content is not verified, and vulnerabilities can be introduced.

Based on the above, when the application analyzes a DApp, any application with the presence of eth_sign and personal_sign methods is considered vulnerable. If eth_signTypedData is detected, versions 1 and 3 are reported as vulnerable, while version 4 undergoes a detailed analysis of the signature content to determine its security.

Considering the results of 17% of the studied applications being vulnerable and an additional 3% potentially insecure, it is concluded that this "market" requires greater integration of security mechanisms and systems. This is primarily because we are dealing with a sector that involves billions of euros and is expected to serve as the foundational support for new banking systems, traceability systems, and more. Therefore, based on these results, the research phase and the initial hypothesis that these applications are less secure compared to other market segments are deemed successful.

8. References

[1] The crypto wallet for Defi, Web3 Dapps and NFTs | MetaMask. (s. f.). https://metamask.io/

[2] DappRadar - The World’s Dapp Store | Blockchain Dapps Ranked. (s. f.). DappRadar. https://dappradar.com/

[3] GitHub: Let’s build from here. (s. f.). GitHub. https://github.com/

[4] Node.js. (s. f.). Node.js. https://nodejs.org/es

[5] CodeQL. (n.d.). https://codeql.github.com/

Índice de la memoria

Índice de ilustraciones

Índice de tablas

Capítulo 1. INTRODUCCIÓN

MOTIVACIÓN DEL PROYECTO

La tendencia en todos los mercados es de, que el poder no recaiga sobre una misma entidad. Esto también ha pasado en el mundo tecnológico y financiero. Debido a esta tendencia, ha aparecido la tecnología blockchain. Esta tecnología permite crear una red descentralizada en la que se validan transacciones (monetarias o no). La principal ventaja es que, al ser un sistema descentralizado, no tenemos esa figura de una entidad reguladora detrás. Esta tecnología es compleja de entender, pero sencillamente es un conjunto de miles de máquinas, que validan y anotan transacciones que ocurren en esta red. Para que no haya fraudes, estas validaciones son comprobadas por muchos nodos y son matemáticamente complejas de obtener. También se mantiene un historial de todas las transacciones.

Desde que esta tecnología se estableció, han aparecido muchas aplicaciones que trabajan sobre el blockchain. Las principales aplicaciones son las relacionadas con la compraventa de criptomonedas. Este tipo de aplicaciones, como puede ser Binance [1], mueven una gran cantidad de capital. En 2022, el periódico El País indicó *"Las criptomonedas movieron unos 60.000 millones de euros en España"* [2].

Como todo mercado en el que hay un gran movimiento de capital y con una tecnología tan reciente, aparecen una gran cantidad de ataques que explotan las vulnerabilidades, con fin de sustraer de forma ilícita dichas monedas. Históricamente, han ocurrido una serie de ataques, como el que ocurrió en una plataforma de compraventa de NFTs (tokens no fungibles), donde se robaron fácilmente 310.00 euros [3]. En la sección del estado del arte se comentará más en profundidad este aspecto de la tecnología.

Este proyecto busca acabar con este tipo de fraude. Para ello, se va a investigar a fondo todos los aspectos relacionados con la seguridad de estas aplicaciones del blockchain, para que el usuario conozca la seguridad de dicha aplicación. Para ello, este proyecto tiene una gran fase

de investigación para acotar los métodos de seguridad que se deben trabajar y determinar el tipo de ataques que se pueden llevar a cabo.

Para finalizar el proyecto, con toda la información recopilada de la fase de investigación y una serie de pruebas realizadas, se analizará el código fuente de una serie de aplicaciones en busca de las vulnerabilidades. Con los resultados obtenidos del análisis estático del código fuente, se creará una herramienta en la que el usuario indique la aplicación que quiere analizar y esta le indique si es segura de usar o no. Como último objetivo, la presentación de un *paper* donde se recojan todos los resultados obtenidos y que se publicará a la comunidad científica. Este último objetivo se llevará a cabo después de la defensa de este proyecto.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para llevar a cabo el correcto desarrollo de este trabajo, se han empleado una serie de herramientas que nos facilitan el trabajo de investigación y la obtención de resultados. Las principales herramientas que se emplean se centran en la parte de investigación, ya que es la fase del proyecto más amplia.

La principal herramienta de este proyecto es Metamask [4]. Esta herramienta proporciona a los usuarios una interfaz y pasarela entre aplicaciones y el blockchain. Contiene una cartera virtual con las criptomonedas y tokens, las cuales se usan para interactuar con las aplicaciones descentralizadas. Uno de los principales usos de esta herramienta es facilitar a los usuarios el uso de las aplicaciones, gestionando todas las funciones de la blockchain. También facilita el trabajo de los desarrolladores de dichas aplicaciones, ya que implementa una serie de mecanismos y lógica. Mucha de esta lógica está disponible de forma de APIs (interfaz de programación de aplicaciones), por lo que los desarrolladores solo se limitan a hacer llamadas a dichas APIs y Metamask gestiona el resto. Esta herramienta es el principal objetivo de investigación del proyecto.

Otra de las herramientas principales del proyecto es CodeQL [5]. Esta es una herramienta de GitHub [6] que permite buscar vulnerabilidades en aplicaciones dentro de una base de datos especificada. Esta herramienta trabaja con análisis estático de código, el cual será explicado detalladamente durante el desarrollo de este trabajo. En este proyecto, se emplea esta herramienta para obtener resultados sobre la seguridad de las aplicaciones. Esta herramienta es muy útil ya que, el código fuente de las aplicaciones que se investigan, no es viable de estudiar de forma manual. Debido a ello, hay una necesidad intrínseca de emplear una herramienta de análisis como es CodeQL.

Otra de las herramientas principales es GitHub [6]. Esta herramienta proporciona un repositorio virtual en internet, donde compartir el código generado para el proyecto. Específicamente, para este proyecto, se trata de una herramienta muy útil, ya que se compone

de varios miembros. Esta herramienta es muy usada en equipos de desarrollo por su facilidad de trabajar sobre el mismo repositorio y gracias a las funciones que proporciona para la gestión de los cambios y versionado.

Por otro lado, y también relacionado con el código, se empleará NodeJS [7]. Esta herramienta proporciona un entorno de desarrollo y ejecución de aplicaciones JavaScript. Principalmente, esta herramienta será usada para la creación de una aplicación descentralizada, en la que se llevarán a cabo pruebas.

Por último, se emplearán otra serie de herramientas conocidas por todos, por lo que no se citarán en detalle. Entre estas herramientas están Google Drive, que nos permite crear documentaciones y compartirlas de forma eficiente con el resto de los miembros. También se emplean códigos de programación como pueden ser JavaScript o TypeScript. Estos dos lenguajes en realidad parten de uno mismo y, como se citará en este proyecto, son los más empleados en la industria de las aplicaciones descentralizadas del blockchain.

Capítulo 3. ESTADO DE LA CUESTIÓN

Desde la aparición de la tecnología blockchain, ha habido una gran serie de ataques y robos de una gran suma de dinero a las aplicaciones descentralizadas. Un ejemplo de estos ataques puede ser el robo de NFTs por un valor de 310.00 euros [3]. Debido a esto hay una necesidad global por parte de la industria de trabajar por la solución y securización de las aplicaciones (ya que la tecnología de blockchain ya es segura de por sí).

Debido a esto, desde hace años se está haciendo un gran esfuerzo por parte de la industria por introducir nuevos mecanismos que acaben con las estafas, robos de transacciones y de las claves privadas de los usuarios. Las principales novedades de seguridad han sido introducidas por las distintas carteras de criptomonedas, como puede ser Metamask. Esta última ha ido variando sus mecanismos de seguridad a la vez que se detectaban los fallos de seguridad. Todos estos mecanismos modernos serán citados a lo largo de este proyecto.

Cada vez más, la responsabilidad recae sobre el usuario, ya que la tecnología actual de estas aplicaciones no es capaz de asegurar que nunca van a ser atacadas. Hay una serie de recomendaciones las cuales el usuario debe de seguir, al igual que una serie de acciones que nunca se deben de realizar, como compartir claves.

Ninguna plataforma y, mucho menos si se mueven grandes cantidades de dinero, debería recaer toda la responsabilidad sobre el usuario. La plataforma debería de aportar toda la seguridad que dispone en sus manos. Debido a esto, con este trabajo se busca mejorar y facilitar al usuario información sobre la seguridad de las aplicaciones. De esta forma (ya que las aplicaciones nunca serán 100% seguras), el usuario podrá comprobar si la aplicación es segura de utilizar (por parte de la plataforma). Se le proporcionará al usuario una herramienta que indicando el código fuente de la aplicación, esta herramienta lo analizará y avisará al usuario sobre el nivel de seguridad de esta aplicación.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Según lo mencionado en las secciones de introducción y del estado del arte, la principal motivación de este proyecto es acabar con las estafas y robos que ocurren en las aplicaciones descentralizadas del blockchain y en las cuales se desperdician millones de dólares al año por este motivo. Un claro ejemplo es uno de los robos más recientes, en el que se robaron al instante 24 millones de euros en criptomonedas [8]. En la era digital en la que nos encontramos no se puede tolerar un "mercado" con tantos fallos de seguridad.

La herramienta que se va a emplear en este trabajo permite al usuario determinar la seguridad de dichas aplicaciones. Con ello, se pretende que el usuario final tenga un mayor conocimiento a priori sobre la aplicación, debido a que la información sobre la seguridad no suele ser pública en muchos casos.

Gracias al estudio que se lleva a cabo en este proyecto, también se generará unos resultados estadísticos los cuales se presentarán a la comunidad científica, ya que son datos representativos de este mercado.

4.2 OBJETIVOS

Como ya se ha citado de forma reiterada, el principal objetivo del proyecto es que el usuario final de las aplicaciones del blockchain, tenga conocimiento sobre la seguridad de la aplicación que va a usar. Para ello, el principal tangible de este proyecto será la creación de una herramienta que sea capaz de determinar el nivel de seguridad de la misma y se la presente al usuario.

El segundo objetivo de este proyecto es que, con la información recabada de analizar una serie de aplicaciones con la herramienta, generar un estudio sobre la seguridad del mercado

de aplicaciones descentralizadas. Este estudio se presentará en formato *paper* a la comunidad científica después de la defensa de este trabajo.

Debido al enfoque de investigación de este proyecto, hay una serie de objetivos que se han ido generando mientras se trabajaba por conseguir el objetivo principal. Uno de estos subobjetivos más importantes es la creación de una documentación detallada sobre los métodos que proporciona Metamask. Aunque este sea un objetivo secundario, es de igual importancia, debido a que la documentación disponible por Metamask y Ethereum es escasa y en muchos casos, no concuerdan entre ellos.

4.3 METODOLOGÍA

Este es un trabajo puramente de investigación como se citó en el anexo B. Se debe de realizar una fase muy extensa de investigación sobre la API de Metamask, estudio de las DApps con más tráfico, De estas aplicaciones también se investiga el lenguaje de programación más empleado y la presencia de algunas APIs críticas para la seguridad de la misma.

Al tratarse de una investigación en profundidad, es necesario también contrastar la información con la documentación disponible de las distintas organizaciones (Metamask, Ethereum, etc). Esto también conlleva que surjan muchos imprevistos o que se encuentren otros objetivos durante el desarrollo de este.

Como ya se ha citado anteriormente, la metodología que se va a seguir para el correcto desarrollo de este proyecto es *Agile*. Esta metodología nos permite ir iterando el producto final en pequeños periodos de tiempo también conocidos como *Sprints*. Esta metodología es clave para un proyecto de desarrollo por lo citado, se va iterando sobre el mismo e incorporando *features* nuevas. Debido a esto, en un trabajo como este, da cabida a imprevistos o nuevos objetivos mientras se realiza la investigación.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

En cuanto a la planificación del proyecto, se ha seguido la misma citada en el anexo B. A continuación, se muestra el cronograma en formato Gantt. En este se puede observar como la gran mayoría de *tasks* son de corta duración y tienen una gran fase de investigación previa. Para finalizar, se acaba con el desarrollo de la herramienta y la evaluación de la misma para concluir con unos datos empíricos que poder compartir con la comunidad científica.

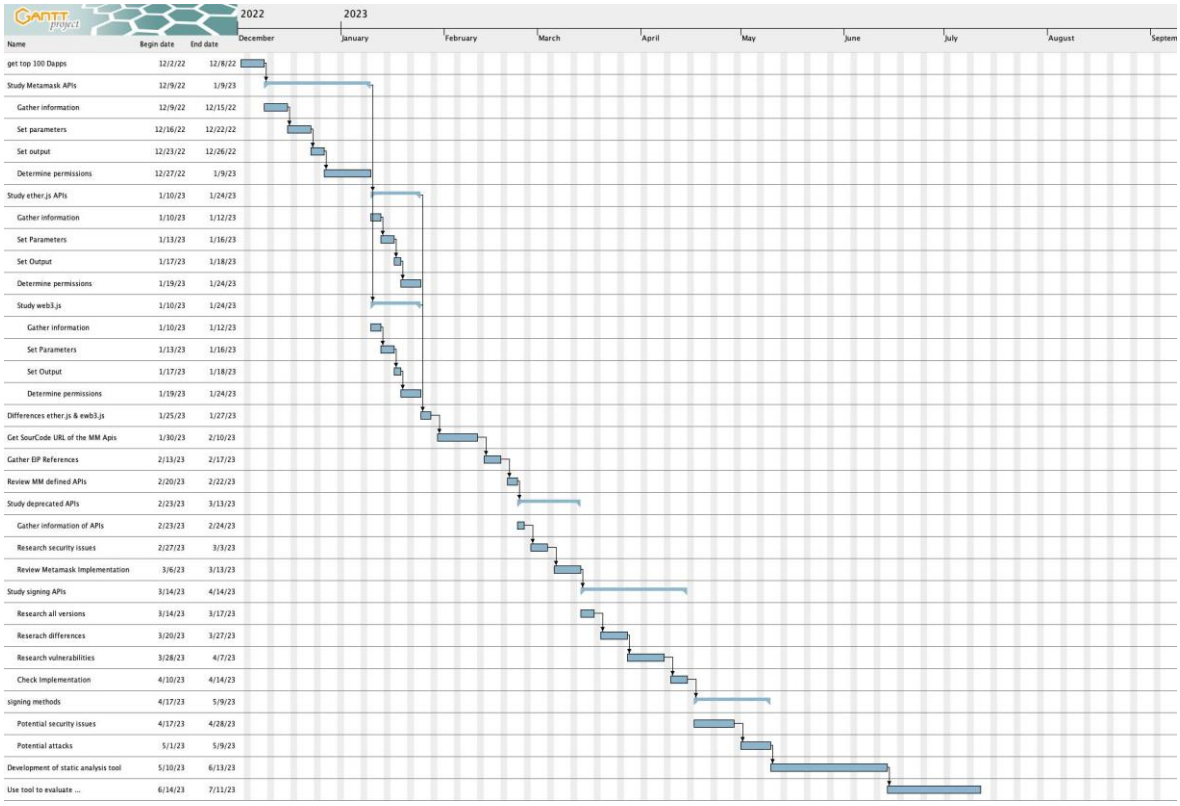


Ilustración 3: Cronograma del proyecto en formato Gantt

En cuanto a la estimación económica, este proyecto no conlleva ningún coste directo. El único coste que se podría llegar a contemplar es el pago de una serie de ingenieros de software para el mantenimiento de la herramienta. También se podría contemplar el coste de puesta en marcha y mantenimiento de un pequeño servidor y página web donde contener la aplicación para su descarga online. Si tenemos en cuenta estos costes, podemos desglosar el coste total anual en:

	Coste estimado	Coste total
Ingeniero de SW	35.000	36.000
Mantenimiento Servidor y Web	1.000	36.000

Tabla 1: Desglose de costes estimados

Capítulo 5. SISTEMA/MODELO DESARROLLADO

En este capítulo se van a tratar todos los temas relacionados con el propio desarrollo del proyecto para alcanzar los resultados. Este se va a estructurar en:

- Análisis de los lenguajes empleados
-

5.1 ANÁLISIS DE LOS LENGUAJES EMPLEADOS

La primera actividad a realizar en el trabajo es determinar el lenguaje de programación más empleado entre las aplicaciones del blockchain. Para ello y empleando la página de DappRadar [9], se ha creado un documento para analizar el lenguaje de las 100 aplicaciones con más tráfico de usuarios en internet. A partir de este documento se obtiene una estadística sobre los lenguajes de programación, el cual concluya que el más usado es TypeScript con un 52%.

Lenguaje de programación	**Porcentaje**
TypeScript	52%
JavaScript	29%
Desconocido	15%
Python	3%
Cadence	1%

Tabla 2: Porcentajes de utilización según leguaje de programación

Como se observa en la tabla, se puede concluir que prácticamente todas las aplicaciones están escritas en TypeScript y JavaScript. Estos dos lenguajes son prácticamente idénticos,

ya que TypeScript está basado en JavaScript y añade una serie de modificaciones. Este dato será muy importante más adelante en el proyecto para cuando se inicie la fase de análisis del código.

5.2 IMPLEMENTACIÓN DEL BLOCKCHAIN

Hay distintos métodos para implementar los métodos del blockchain en una aplicación. Todos estos tienen sus ventajas y desventajas y se comentarán en esta sección.

La primera forma es implementar todos los métodos de una cartera del blockchain (Metamask, Binance, etc). Este método requiere que cada uno de los métodos que se definen en las documentaciones de estas carteras, sean implementados. Si se emplea esta opción, se nos ofrece el "poder" de poder controlar todos los aspectos de las aplicaciones, siendo los desarrolladores los que limitan la funcionabilidad de esta. Por otro lado, el principal problema es que, si queremos que la aplicación soporte más de una *wallet*, hay que programar la aplicación con tantas versiones como *wallets*, lo cual no es muy eficiente y mantenible.

La otra opción disponible es la programación de todos estos métodos a través de las librerías Web3.js [10] o Ether.js [11]. Estas dos son las librerías principales estudiadas en este trabajo, ya que se investiga únicamente aplicaciones que trabajan sobre el blockchain de Ethereum. Todas las funciones citadas que se debían implementar de las *wallets*, también se deben de implementar con estas librerías. La diferencia es que, no se implementarán las de cada cartera en específico, si no la función genérica que proporcionan las librerías. Debido a esta diferencia, se proporciona mucha flexibilidad a la hora de la programación de la aplicación ya que, estas librerías aceptan el uso de cualquier cartera compatible. Es decir, se programa la aplicación con los métodos genéricos de las librerías y se le permite al usuario conectar su cartera.

Otra ventaja y la cual es clave para el desarrollo de este proyecto es que, cuando se ha programado la aplicación con estas librerías y se conecta una cartera, los métodos de las

librerías traspasan la responsabilidad a la cartera. Es decir, de cara al usuario es como si se hubiese programado con la cartera directamente. Esto también implica a los métodos que requieren permisos del usuario, como se citará a posteriori. Gracias a estas librerías se consigue que, con una sola versión de la aplicación, se soporten varias carteras. A continuación, se muestra un pequeño ejemplo de un mismo método en Web3.js comparado con Metamask (este último solo muestra el contenido de la petición API). Este método se usa para solicitar información para saber si se está minando criptomonedas o no:

Web3.js	Metamask
`Web3.eth.isMining([callback])`	`{ "jsonrpc": "2.0", "method": "eth_mining", "params": [], "id": 0 }`

Tabla 3: Comparación Web3.js y Metamask

Un ejemplo visual de cómo son las aplicaciones que utilizan estas librerías pueden ser Uniswap [12], la cual es la DApp con más tráfico de internet. Esta aplicación proporciona a los usuarios una plataforma donde intercambiar, comprar o vender criptomonedas. Como se puede observar en la siguiente imagen, esta plataforma acepta conectar distintas carteras y, como ya se ha citado, esto se consigue con las librerías Ether.js y Web3.js:

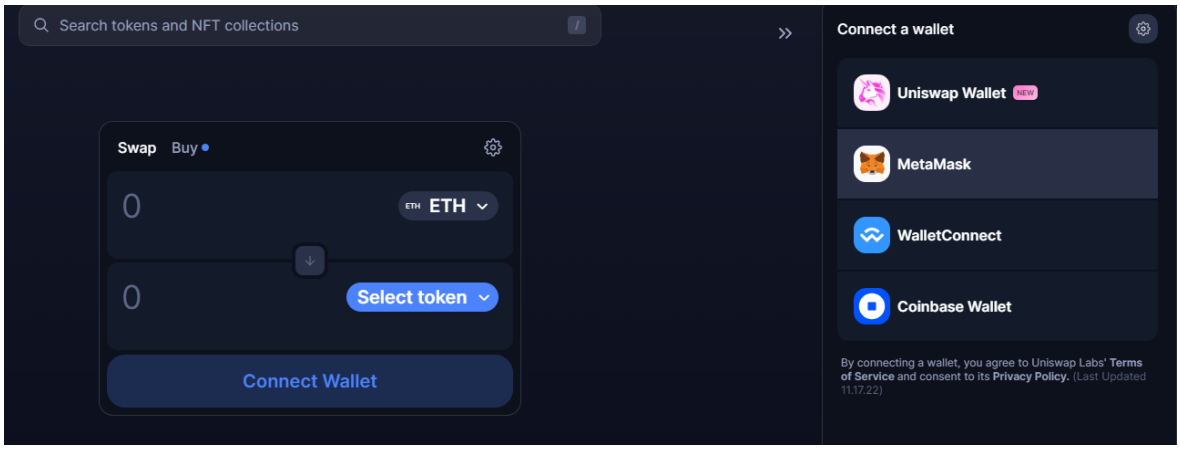


Ilustración 4: Ejemplo de múltiples carteras. Uniswap.

En el caso de esta aplicación, si se investiga el código fuente [13], se observa que la librería empleada para la implementación de los métodos del blockchain es Web3.

Una vez visto que, independientemente del lenguaje de implementación del blockchain, la lógica que predomina es el de la cartera, se procederá a estudiar los métodos proporcionados por Metamask.

5.3 API DE METAMASK

Esta es una de las fases más importantes del proyecto, ya que al estudiar los métodos que proporciona Metamask, también investigaremos cuales de ellos están relacionados con la seguridad.

Para la obtención de la información, se ha empleado la documentación que Metamask pone a disposición de usuarios y desarrolladores. Esta documentación es escasa y poco precisa, como se citará más adelante. Estas dos fuentes de documentación son, la página oficial de la documentación de Metamask [14] y una página oficial [15] para probar los distintos métodos. En cuanto a la primera documentación, es muy escasa, ya que solo recoge algunos métodos muy selectos. Entre estos métodos están los restringidos, los cuales son los métodos que requieren permisos definidos en el EIP-2255 [16] (los documentos tipo EIP serán comentado más adelante). Debido a esto, los métodos de esta documentación son muy limitados, e incluso, muchos no están bien desarrollados. Por otro lado, la página de pruebas de Metamask contiene casi todos los métodos disponibles. En esta página también se explica brevemente la funcionalidad de los métodos, estructura de los parámetros de entrada y de la respuesta, así como cuales de ellos son obligatorios y cuales opcionales.

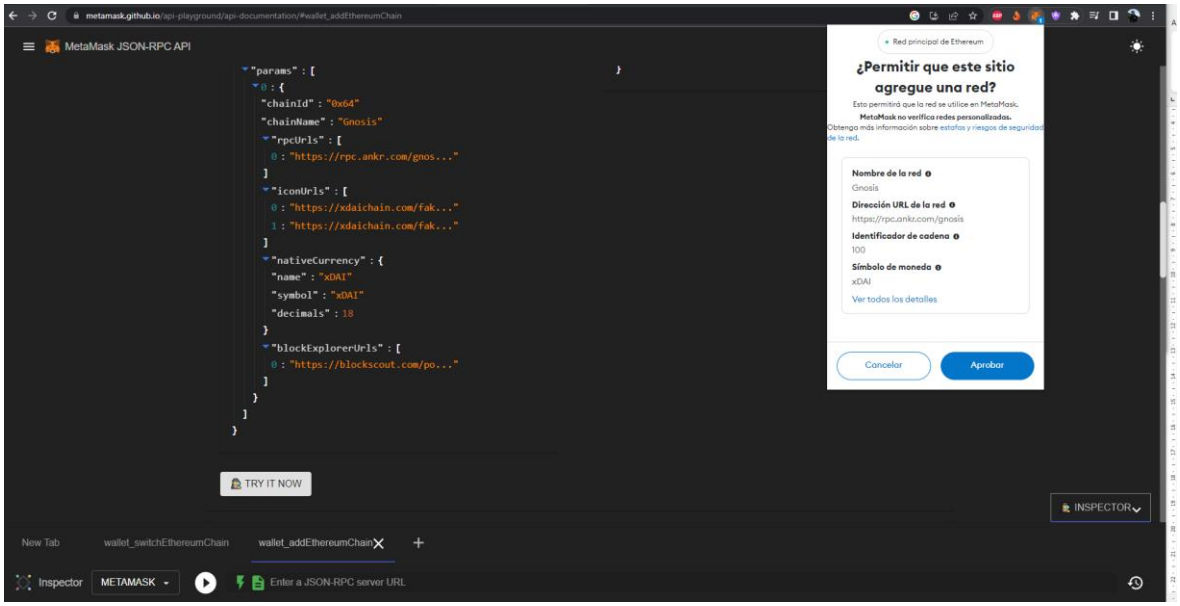


Ilustración 5: Ejemplo de prueba en Metamask Playground [15]

Con la información recabada de estas dos documentaciones, se ha creado un documento tipo Excel que contienen todos los métodos que proporciona Metamask (y los heredados de Ethereum), una breve descripción, los parámetros obligatorios, los parámetros opcionales, si hace falta confirmación por parte del usuario y el formato de la respuesta.

	A	B	C	D	E	F	G	H	I
1	API name	Brief definition	Required Parameters	Optional Parameters	Permission/ Confirmation	Return (Doc)	permissions comment	Differences	Location of Source Code
2	eth_requestAccounts	Connect dapp to MM wallet			Permission Needed	string[] / error code	Permission Needed		Github Repo
3	wallet_getPermissions	get the users permissions				id: String @context: Array (String) invoker: String caveats: Array (Object){ - Type: string - Value: ? - name: String}		EIP specifies other result: - invoker: string - parentCapability: string - caveats: Array Object{ - Type: String - value: string}	GitHub Repo
4	wallet_requestPermissions	Requests additional permission from the user	eth_accounts: Object		confirmation needed	id: String, @context: Array (string), invoker: String, caveats: Array (object)	First selecting the account, then connect or cancel confirmation needed	Minor: EIP does not specify the result when approved only specifies null when rejected. Return in the EIP isay it must be a requestedPermission objects but it doesnt specify the parameters.	Definition in the controller https://github.com/MetaMask/metamask-extension/blob/develop/app/scripts/metamask-controller.js Use https://github.com/MetaMask/metamask-extension/tree/develop/app/scripts/controllers/permissions
5	eth_accounts	Returns a list of addresses for the accounts owned(allows you to access the user's Ethereum address(es))				string array / null			GitHub Repos
6	eth_decrypt [DEPRECATED]	Requests that MetaMask decrypts the given encryoted message	EncryptedMessage: String, Address: String		Confirmation needed (Add token or cancel)	id: String, @context: Array (String), invoker: String, caveats: Array (Object)	The problem with this one is that the same private key is used for all the methods that needs encryption and that increases exponentionally the chance of the private key to get stolen.	In the github repo they also add: - version - nonce - ephemPyblicKey - cipertext	GitHub Repo

Ilustración 6: Imagen de la documentación creada de Metamask

Como se puede observar se han insertado también las direcciones al repositorio de GitHub de Metamask [17] y algunas columnas extra que se comentarán más adelante. El principal objetivo de esta fase es documentar de forma clara y sencilla todos los métodos que se

proporcionan en la API de Metamask. También en esta fase es muy importante determinar cuál de estos métodos son los que requieren permisos o confirmación por parte del usuario. Como ya se ha citado, estos son los métodos objetivo del estudio de este proyecto, ya que esos métodos son los que hacen que una aplicación sea segura o no.

Para determinar que métodos requieren permisos o confirmación, primero se investiga en el *Playground* de Metamask [15] donde, al ejecutar la prueba de ese método, si aparece el *popup* de la extensión de Metamask, se puede determinar si es necesaria la confirmación. Un ejemplo de ello es el método *wallet_switchEthereumChain* que, al ejecutarlo, nos salta la pestaña de Metamask para aprobar el cambio de la cadena.

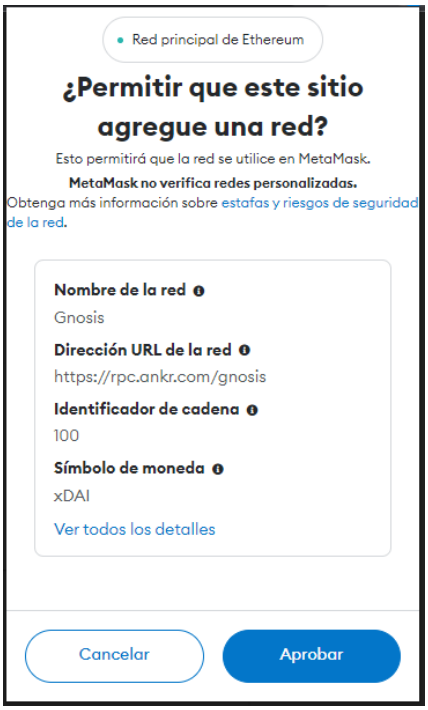


Ilustración 7: Popup de confirmación wallet_switchEthereumChain

Hay otros de estos métodos los cuales no es tan sencillo determinar si necesitan confirmación o no. En estos casos, se debe analizar la implementación de dichos métodos en el repositorio de GitHub de Metamask [17]. Por ello, como se citó anteriormente, a la documentación se añadió la dirección web de donde están implementados dichos métodos dentro del código fuente y especialmente la documentación del EIP.

El EIP (*Ethereum Improvement Proposals*) es la documentación detrás de Ethereum como "organización". En los EIP se encuentran sobre todo los métodos que se implementan en Ethereum así como las buenas prácticas, ejemplos y en algunos casos, el código completo.

Para determinar si un método requiere permisos por parte del usuario a partir del EIP, se debe de leer con detalle toda la especificación. En ella puede aparecer mencionado que el usuario debe de confirmar o incluso, puede que se tenga que determinar a partir del código. En muchos casos, viendo como se genera el resultado de del método. De esta última forma, el resultado muchas veces se especifica cuando:

- El proceso es correcto y el usuario acepta.
- El usuario rechaza la petición de permisos.
- Otros casos.

Un ejemplo claro de esto es el método *wallet_addEthereumChain*. Al consultar el EIP número 3085 [18], que es el relacionado con este método, se explica tanto en texto como en la sección del resultado, que el usuario debe aceptar o rechazar y en cuanto al resultado del método, especifica cómo debe ser en caso de rechazo o aceptación.

5.4 API ETHER.JS Y WEB3.JS

Al igual que ocurre con Metamask, las librerías de Ether y Web3 también porporcionan una documentación de los métodos que se ofrecen a los programadores. Como ya se ha citado anteriormente, en las aplicaciones programadas con estas librerías, realmente la responsabilidad sigue recayendo sobre la lógica de la cartera que el usuario connecte. Debido a esto, la lógica y complejidad del desarrollo de aplicaciones con Ether.js o Web3.js, es mucho menor.

La documentación de estas dos librerías es mucho más extensa y mucho más fácil de entender de cara a los programadores (esto también se debe a que son menos complejas). Cono se ha mostrado anteriormente, el aspecto de estos métodos no son como los de Metamask (que son más peticiones API), sino

que son más parecidos a una función o método de un lenguaje de programación orientado a objetos.

En este proyecto no se ha estudiado la seguridad de estas dos librerías, si no que el trabajo se centra en las APIs de Metamask. Debido a que se puede vincular la cartera de Metamask con aplicaciones desarrolladas en Ether y Web3, con estudiar Metamask, estudiamos también estas dos últimas librerías. Debido a esto, en este apartado, se ha documentado método a método, al igual que en la sección 5.3, con parámetros obligatorios, opcionales, retorno del método, etc. Lo más importante de esta sección es documentar los métodos que corresponden a los de Metamask. Por ello, en el documento tipo Excel, se han creado tres hojas de cálculo (una para Metamask, otra para Ether.js y otra para Web3.js), y los números de columna concuerdan con los métodos. Es decir, el método de metamask en la fila X, corresponde con los mismos métodos, pero de EtherJS y Web3JS en la misma fila X.

Como se puede observar en las siguientes ilustraciones, la estructura de la documentación sigue el mismo formato que el visto en la sección 5.3.

18	provider.getBlockNumber()	Returns the block number (or height) of the most recently mined block.			Promise< number >							
19	provider.call()	Returns the result of executing the transaction, using call. A call does not require any ether, but cannot change any state. This is useful for calling getters on Contracts.	transaction: Transaction	blockTag: BlockTag	Promise< string< DataHexString > >							
20	signer.getChainId()	Returns the chain ID this wallet is connected to.			Promise< number >							
21												
22	provider.estimateGas()	Returns an estimate of the amount of gas that would be required to submit transaction to the network.	transaction: Transaction		Promise< BigNumber >		signer.estimateGas()	Returns the result of estimating the cost to send the transactionRequest, with this account address being used as the from field.		transactionRequest: TransactionRequest		Promise< BigNumber >

Ilustración 8: Documentación creada de Ether.js

18	web3.eth.getBlockNumber		web3Context; web3Context, returnFormat: returnFormat				Promise<NuberTypes[ReturnFormat[number]]>
19							
20	web3.eth.getChainId		web3Context: web3Context, returnFormat: returnFormat				Promise<NuberTypes[ReturnFormat[number]]>
21	web3.eth.gerCoinbase		web3Context: web3Context				Promise<string>
22	web3.eth.estimateGas		**Transaction**: From (string), to (string), value (string), nonce (string), type (string)				Promise<NuberTypes[ReturnFormat[number]]>
23	web3.eth.getGasPrice		web3Context: web3Context, returnFormat: returnFormat				promise<NumberTypes[ReturnFormat[number]]>
24	web3.eth.getBalance		web3Context: web3Context, address: String, blockNumerOrTag, returnFormat: returnForma				promise<NumberTypes[ReturnFormat[number]]>

Ilustración 9: Documentación creada de Web3.js

5.5 PROBLEMAS EN LA DOCUMENTACIÓN

Durante los procesos de documentación de los distintos métodos (específicamente los de Metamask), ha surgido un problema no esperado y que es digno de un estudio aparte. Al crear la documentación citada en el apartado 5.3, se iba contrastando la información, que hay que recalcar que era escasa, de la documentación de Metamask, con la extensa documentación del EIP. Este tipo de "choques" de las dos documentaciones, normalmente afecta a la seguridad de las aplicaciones.

Durante este proceso, se ha concluido que, en muchos casos, lo que se implementa no es lo que se recomienda. Un claro ejemplo de esto es el método *wallet_addEthereumChain*. En este método, Metamask y el EIP especifican una serie de parámetros obligatorios y opcionales. En concreto este ejemplo es muy bueno ya que, en el EIP, estos parámetros opcionales, cita que deberían de ser obligatorios (aunque no estén especificados como tal). Esto se debe a que esto se trata de buenas prácticas, lo que no es obligatorio (aunque prácticamente lo sea).

Esto ocurre en muchos métodos. Debido a esto, el primer resultado que se crea con este proyecto es la documentación creada durante la fase de investigación. Todas estas diferencias se han añadido a la hoja de Excel donde están contenidos todos los métodos de Metamask. Por ejemplo, se puede ver en la siguiente imagen un ejemplo de la documentación que se ha creado. Este método es *wallet_watchAsset*, donde Metamask especifica todos los parámetros como opcionales, mientras que en el EIP (en rojo), especifica dos parámetros como obligatorios:

7	wallet_watchAsset		**Type:** string **options**: (object){ address: String, symbol: String, decimals: number, image: String}	confirmation needed	boolean	Confirmation needed (Add token or cancel)	The EIP has different Required Parameters. This API has two main parameters: - **Type**: String (required in EIP but not in metamask). - **Options** (object){ - address: String (required in EIP but not in metamask) - symbol: String (optional in both docs) - decimals: number (optional in both docs) - image: String (optional in both docs)	GitHub Repo

Ilustración 10: Ejemplo de colisión de la documentación en wallet_watchAsset

5.6 MÉTODOS DE FIRMA

En el blockchain es necesaria la implementación de distintos métodos de firma. Esto se debe a que, por seguridad, toda la información que se quiere transmitir a través de esta cadena de bloques se firma por la seguridad e integridad de los datos, y para confirmar que el emisor es el esperado.

Con el paso de los años, se han ido descubriendo nuevas formas más seguras de firmas y lo mismo ha ocurrido. Metamask [19] tiene distintos métodos de firma que se han ido implementando con el tiempo. Los principales son:

- *Eth_sign*
- *Personal_sign*
- *Eth_sigTypedData_v4*

5.6.1 *ETH_SIGN*

El primer método es el menos seguro, ya que en el *popup* que se le presenta al usuario, el mensaje de la firma no es legible. Por ello, muchos ataques se basaban en presentarle al usuario esta firma, la cual como no se puede leer, lo firmaban pensando que es una transacción legítima, cuando en realidad la aplicación les hace firmar otra cosa. Debido en el momento en el que se inició este estudio, este método no se recomendaba. Ahora en el momento que se está realizando este documento, el propio Metamask a catalogado este método de firma como deprecado. Para saber más de estos métodos, ver Anexo I. Como se aprecia en la siguiente imagen, el mensaje a firmar es una cadena hexadecimal que surge de realizar el hash de la firma, por lo que el usuario no lo puede entender.

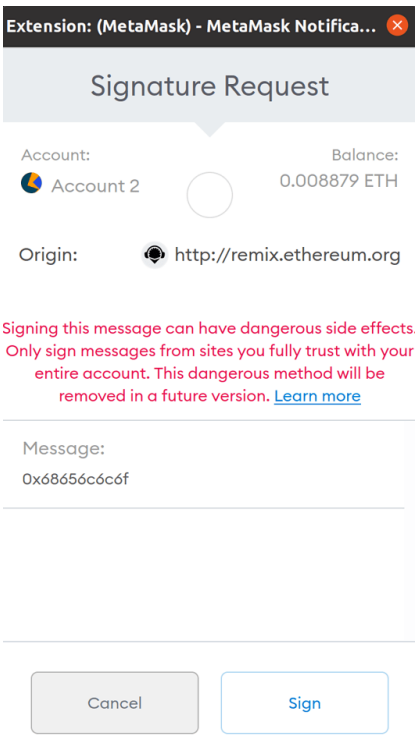


Ilustración 11: Ejemplo de eth_sign

Debido a la inseguridad de esto, Metamask en su propia aplicación, en los ajustes avanzados, ahora permite desactivar las peticiones a este método. Gracias a esto y, a que se disponen de los otros métodos que se citarán a continuación, Metamask no dejará a las aplicaciones usar este tipo de firma [20]. En general, las aplicaciones ya usan los otros métodos más modernos, por lo que la aplicación funcionará sin problema. Aun así, muchas aplicaciones siguen empleando el método *eth_sign* como método de firma por lo que, en caso de tenerlo desactivado, estas aplicaciones no dispondrían de toda la lógica, haciendo que parte de su funcionamiento sea inutilizado.

5.6.2 PERSONAL_SIGN

Este fue el método que Metamask introdujo para solucionar que el mensaje de la firma no sea legible. Esta función es la forma más fácil de implementar. El principal problema de este método y el siguiente es que, como el mensaje si es legible, hace que sea menos eficiente para ser procesado en el blockchain.

Este método en específico es poco eficiente y por ello surgió el que se comentará en la siguiente sección. Aun así, muchas aplicaciones todavía usan este método debido a que es muy simple. A continuación, se muestra el aspecto del mensaje que se le presenta al usuario al ejecutar este método:

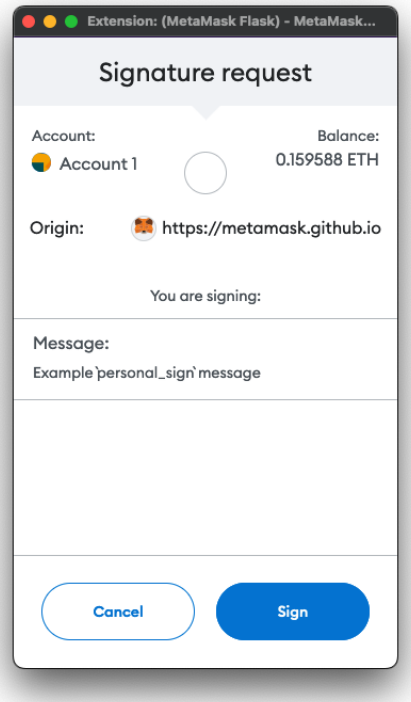


Ilustración 12: Ejemplo de personal_sign

5.6.3 ETH_SIGNTYPEDDATA_V4

Este es el método más moderno implementado en Metamask. Este es el método en teoría más seguro y flexible. Esto último se debe a que, permite varias direcciones de envío, varias de destino y múltiples parámetros. Más adelante, se comentará los problemas que conlleva tener tanta flexibilidad. A continuación, se muestra el aspecto del *popup*:

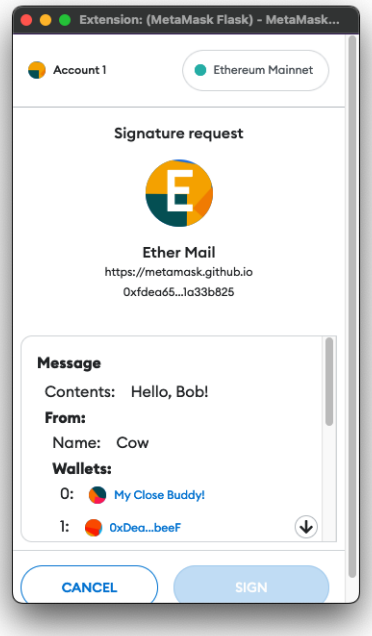


Ilustración 13: Ejemplo de eth_signTypedData_v4

Como se puede observar, en la confirmación se incluye unos datos estructurados (from, wallets, etc). Este método es el que implementa de forma más eficiente el hecho de que sea legible por parte del usuario. Esto conlleva que el código de implementación es más complejo.

5.6.3.1 Estudio de la implementación

Debido a que este método es el más moderno y tiene más seguridad de los tres, se procederá a estudiar en detalle su implementación y posibles problemas de seguridad. Para ello, primero se empieza por estudiar cómo debe implementarse y definirse.

Para entender en profundidad cómo funciona y como se implementa este método, se debe consultar el EIP-712 [21][22]. En estas documentaciones hay 4 pasos muy concretos que se van a comentar a continuación:

1. Diseñar la estructura de los datos: En este paso, se va a definir qué datos se van a firmar. Este método requiere que todos los datos estén estructurados y con la

definición de sus tipos. La mejor forma de entender este concepto es viendo un ejemplo. A continuación, se muestra un archivo JSON (solo la parte correspondiente a este punto) que contiene todo lo necesario para poder ser usado en el método *eth_signTypedData_v4*.

```
"message": {
    "contents": "Hello, Bob!",
    "from": {
      "name": "Cow",
      "wallets": [
        "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826",
        "0xDeaDbeefdEAdbeefdEadbEEFdeadbeEFdEaDbeeF"
      ]
    },
    "to": [
      {
        "name": "Bob",
        "wallets": [
          "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB",
          "0xB0BdaBea57B0BDABeA57b0bdABEA57b0BDabEa57",
          "0xB0B0b0b0b0b0B000000000000000000000000000"
        ]
      }
    ]
  },
```

Este código muestra la creación de la estructura de los datos y su definición (asignación de valores). También hay que definir los tipos de estas estructuras y sus variables, definido en *types*:

```
"types": {
    "Group": [
      {
        "name": "name",
        "type": "string"
      },
      {
        "name": "members",
        "type": "Person[]"
      }
    ],
    "Mail": [
      {
        "name": "from",
        "type": "Person"
      },
```

```
      {
        "name": "to",
        "type": "Person[]"
      },
      {
        "name": "contents",
        "type": "string"
      }
    ],
    "Person": [
      {
        "name": "name",
        "type": "string"
      },
      {
        "name": "wallets",
        "type": "address[]"
      }
    ]
```

La asignación de los tipos es muy importante por lo que se explicará más adelante. También es muy importante ver que, todas las variables que se han establecido tienen un tipo de variable asignada en la estructura *Types*.

2. En el segundo paso hay que diseñar el *domain separator*. Este es una estructura de datos, al igual que los del paso 1, que sirve para diferenciar la función de firma de dicha aplicación con otras que puedan existir (evita conflictos de funciones similares entre aplicaciones). A continuación, se muestra la creación de la estructura del separador de dominio, sus asignaciones de valores y la estructura y definición de los tipos:

```
"domain": {
    "chainId": "0x1",
    "name": "Ether Mail",
    "verifyingContract": "0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCccccccccC",
    "version": "1"
  }
  "types": {
    "EIP712Domain": [
      {
        "name": "name",
```

```
        "type": "string"
      },
      {
        "name": "version",
        "type": "string"
      },
      {
        "name": "chainId",
        "type": "uint256"
      },
      {
        "name": "verifyingContract",
        "type": "address"
      }
    ],
  }
```

Al igual que antes, es muy importante que todas las variables usadas en el separador estén bien definidas con sus tipos. En el EIP-712 se especifica que, este separador debe de contener las variables:

- ChainID: este parámetro contiene el identificador de la red en la que trabajan los contratos.
- Name: Nombre que se le da al separador.
- VerifyingContract: este contiene la dirección del contrato Solidity que va a encargarse de certificar y validar las firmas de este método.
- Version: Version empleada.
- Salt: coste de la firma en tipo salt (esto es opcional).

3. En este paso se define y se programa el código encargado de realizar la firma.
4. Por último, se crea el código encargado de verificar el contrato y autentificar el código de firma.

Otro campo que se añade en el JSON que se firma es el *primaryType*. Este campo determina, de la lista de las variables, cual de ella es la principal. La estructura que se define como primaria, es la que se muestra en la parte superior (y principal) del *popup* que se le presenta al usuario. Debido a esto, es muy importante esta clase, ya que es la

información que el usuario usará para determinar si todo está correcto y proceder a su firma.

5.6.4 CREACIÓN DE LA BASE DE DATOS

Uno de los objetivos principales de este trabajo es realizar un estudio en detalle sobre la seguridad de las aplicaciones reales que los usuarios usan alrededor del mundo todos los días. Para poder llevar a cabo este estudio, primero se debe de recabar información de estas. Por ello, para realizar el estudio sobre los tres métodos mencionados, ha habido la necesidad de crear una base de datos de la cual partir.

Esta base de datos se ha creado de forma manual, obteniendo las aplicaciones más usadas de internet. Para obtener este listado, se ha empleado la página de DappRadar [9]. Como ya se ha citado anteriormente, esta página nos proporciona un listado con las aplicaciones descentralizadas que más tráfico generan en internet. Este dato es muy importante, ya que las que más tráfico generan, son las más usadas (no hay aplicaciones que aporten una lógica tan distinta como para aportar mucho tráfico con pocos usuarios).

Gracias a esta página, se han obtenido alrededor de 800 aplicaciones. Sobre todas las aplicaciones obtenidas para el estudio, se ha anotado en un documento tipo Excel sus repositorios de código (GitHub), y la presencia de estos tres métodos. Para esto último, se ha tenido que analizar todo el código fuente de cada una de las aplicaciones en busca de estos tres métodos de uno en uno. En esta fase quiero recalcar el gran trabajo que hay detrás de esta fase, la cual es muy importante para el correcto desarrollo del proyecto, pero que no es un objetivo. A continuación, se muestra un pantallazo del aspecto de esta base de datos:

	Dapp name	Language	Source code link	Presence of "eth_signTypedData"	Presence of "eth_sign"	Presence of "personal_sign"		Smart contract
2	Uniswasp V3	TypeScript	https://github.com/Uniswap/conedison	32 (usage 80)	Not found	Not found		https://github.com/Uniswap/v
3	Aave Protocol	TypeScript	https://github.com/aave/protocol-v2	276	Not found	Not found		https://github.com/aave/protc
4	Curve	TypeScript	https://github.com/curvefi/curve-js	Not found	Not found	Not found		https://github.com/curvefi/cur
5	PancakeSwap	TypeScript	https://github.com/pancakeswap/pancake-frontend	213	Not found	Not found		https://github.com/pancakesv
6	DODO	TypeScript	https://github.com/DODOEX/dodo-v3	Not found	Not found	Not found		https://github.com/DODOEX/
7	Compound	TypeScript	https://github.com/compound-finance/compound-js	138	Not found	Not found		https://github.com/compound
8	Venus Protocol	TypeScript	https://github.com/VenusProtocol/venus-js	138	Not found	Not found		https://github.com/VenusProt
9	Uniswasp V2	TypeScript	https://github.com/Uniswap/conedison	32 (usage 80)	Not found	Not found	Difference betwe	https://github.com/Uniswap/v

Ilustración 14: Ejemplo de la documentación de aplicaciones y los métodos

En las columnas de los métodos, los números azules que apareces en la tabla son el número de línea donde aparecen dichos métodos. Como estos repositorios tienen muchos archivos, y en muchos pueden aparecer, cada número tiene un hipervínculo al archivo.

Para la búsqueda de las líneas en las que aparece cada método, se empezó realizando una búsqueda con VSCode. Debido al tamaño de la base de datos, se ha creado un programa Shell que, recibe como entrada el nombre del archivo que contiene una lista de todas las direcciones de los repositorios. Posteriormente, pregunta por el nombre del archivo de salida, que contendrá el archivo y línea en la que se han detectado los métodos. Por último, pregunta si la petición de buscar los métodos se realiza en el contrato de verificación (Solidity) o en el código fuente, como es este caso.

Esta herramienta se puede encontrar en el repositorio creado para este proyecto [23]. Para que esta herramienta funcione de forma correcta, también se debe de instalar una dependencia de GitHub [24].

5.7 ANÁLISIS DE SEGURIDAD DE ETH_SIGNTYPEDDATA_V4

El siguiente proceso es estudiar la seguridad de los métodos. En este proyecto se ha decidido centrar la atención en este método. Esto se debe a que este método representa un gran porcentaje de las funciones que implementan las aplicaciones. Para el estudio, lo más importante es la implementación de este método. Ya que se ha creado una base de datos detectando en que aplicaciones y qué líneas y archivos se encuentra este método, hay que determinar qué es lo que puede hacer este método inseguro.

El código y la implementación que se va a estudiar es la mostrada en la sección anterior. Ese código mostrado es el código de ejemplo propuesto por Metamask en su documentación. Este ejemplo, como comentado, contiene varias carteras de destino, un mensaje que se muestra al usuario, etc. Esto es gracias a que, este método habilita una flexibilidad que en el resto de los métodos era imposible. Este método también corrige errores ya conocidos de los otros métodos, como implementar una misma curva de firma (una misma clave de firma)

para varios propósitos no relacionados. Este era el principal problema de *eth_sign* y que se corrige con el separador de dominio. Este separador, permite que el método de firma se defina para un dominio en concreto y corrige la posibilidad de haber varios métodos igualmente definidos intenten ejecutarse a la vez y genere conflictos.

El principal beneficio de este método es la flexibilidad que aporta. Para obtener esta flexibilidad, el método define una serie de parámetros obligatorios y luego ya permite que se creen estructuras para crear la firma deseada. La estructura mínima e indispensable para que la firma sea exitosa es la siguiente:

```
{
  "domain": {
  },
  "message": {
  },
  "primaryType": "Mail",
  "types": {
    "Mail": [
    ]
  }
}
```

En el código anterior, se muestra como la firma es exitosa con un dominio nulo, un mensaje nulo y en *types* y *primaryType* tienen que estar definidos a un tipo, en este caso es Mail, pero podría tener cualquier otro nombre.

Para comprobar que esta es la estructura mínima que genera una firma correcta, se ha hecho uso de la aplicación descentralizada de pruebas que se ha creado para este proyecto [25]. En esta herramienta se debe primero conectar la cartera de Metamask, y posteriormente seleccionar la versión del método *eth_signTypedData* (este trabajo se centra en la versión 4, ya que es la última y la más segura). Después de seleccionar dicha versión, en el cuerpo de la página hay un cuadro de texto donde se debe de introducir el JSON que queremos firmar. En este caso, se parte del código completo mostrado en la sección anterior y se van eliminando estructuras de datos hasta llegar al código mostrado anteriormente.

La aplicación es clave para determinar esta estructura mínima que genera una firma exitosa. Esta aplicación, cuando se intenta firmar, genera un *popup* donde se le informa al usuario de toda la información de la firma y el propio usuario puede aceptar o denegar esta firma.

Para determinar que la firma es exitosa, la aplicación, por debajo del cuadro de texto, después de que el usuario acepte la firma, se muestra la dirección del contrato que lo ha verificado y si ha sido exitoso o no. Gracias a esta lógica que se ha añadido, es fácil de determinar cuál es el contenido mínimo que se puede firmar con este método (partiendo de un contenido completo e ir quitando datos hasta que no se puede reducir más). A continuación, se muestra el mensaje de firma exitosa con el código mínimo:

Ilustración 15: Firma exitosa con mínimo contenido [25]

Ahora que ya se ha determinado la estructura mínima que se puede firmar con resultado exitoso, se debe ejercer de crítico. Un método que a priori es seguro debería de analizar los contenidos que se firman y requerir una serie de mecanismos de verificación de este. Como se puede observar del contenido mínimo, no se realiza ningún tipo de comprobación previa y recae la responsabilidad de fiarse en el usuario. Esto se debe a que este método sí muestra al usuario el contenido, como se ejemplificó en secciones anteriores.

Esta no es una práctica muy recomendable, ya que la gran mayoría de problemas de seguridad siempre vienen por la vía humana. Este método, como se puede observar, no implementa ningún tipo de validación, ni si quiera en el separador de dominio.

El único tipo de verificación que se ha detectado en esta fase es, cuando se usa una variable y no se define el tipo. El método comprueba que todas las variables que se han usado en las distintas estructuras de datos lleven asignadas un tipo de variable en la sección de *types*. Esto se debe a que también este tipo de dato debe concordar con el asignado en el contrato que verifica la firma y, en caso de no estar definido, el contrato deniega la verificación de la firma. A continuación, se muestra un ejemplo:

```
{
  "domain": {
    "chainId": "0x1"
  },
  "message": {
    }
  },
  "primaryType": "Mail",
  "types": {
    "Mail": [
    ]
  }
}
```

Como se observa, en el dominio se ha empleado el uso de la variable chainId, se le ha asignado un valor, pero en la sección de Types no se ha declarado la estructura EIP712Domain y la variable chainId. Debido a esto, cuando se trata de firmar este contenido, el contrato que se encarga de verificarlo procederá a comunicar un error.

Durante la investigación de este método se concluye que, el método en sí, cuando es creado de forma completa (con todo el contenido), es bastante seguro. Por el otro lado, el método no incorpora mecanismos de verificación del contenido. Esto quiere decir que la firma puede ser válida y completamente correcta de cara al blockchain, pero que el contenido no lo sea. Esto puede generar muchos errores en la red, ya que a un usuario le puede llegar que una transacción se ha validado y que todo es correcto, pero en realidad esa transacción iba vacía.

Debido a esto, estos métodos deberían establecer unos límites a la flexibilidad que ofrecen. Como ejemplo, debería de especificar que se valide el dominio, obligando a que este contenga todos los parámetros comentados en el EIP-712. Al añadir esto, se no permite al programador y, sobre todo, a los posibles atacantes, que introduzcan contenidos maliciosos a la firma.

En cuanto al contenido relacionado con la parte definida por la aplicación descentralizada, este contenido es muy complicado de limitar y validar, ya que cada aplicación tiene unos contenidos muy distintos. Aun así, se deberían de establecer unos parámetros que siempre deben de estar, como la cartera de origen y la de destino, así como el mensaje legible para que el usuario confirme estos contenidos. Esto, debería de ser obligatorio y con validaciones.

Debido a todo lo mencionado, se concluye que, aunque este método sea el más seguro implementado por Metamask, que podrían generar problemas y complejidades innecesarias. Todo esto, hace que los desarrolladores opten por métodos más simples, como puede ser e*th_sign*, el cual está deprecado y se ha demostrado que es inseguro.

5.8 CODEQL

Esta es una herramienta desarrollada por el propio GitHub y que permite realizar peticiones tipo *query* sobre las bases de datos de repositorios. Esta base de datos se compondrá de todos los repositorios de GitHub que se quieran analizar. Esta herramienta es muy potente debido a la gran flexibilidad que estas peticiones proporcionan al desarrollador. Las peticiones tienen la particularidad de que son programables. Gracias a ello, se puede llegar a un nivel de detalle en estas.

Otro punto que cabe destacar es que, a pesar de que esta herramienta pertenece a una gran compañía como Github, la documentación es casi inexistente. Durante el desarrollo del proyecto, se ha llegado a dar con varios problemas, los cuales la documentación no era para nada útil. Esto pasa desde el primer contacto con la aplicación, en la que se puede ver que la guía de instalación también es bastante escasa y poco precisa. Debido a esto, esta parte del

proyecto, a pesar de ser importante, se ha tenido que llegar a un punto entre complejidad y utilidad.

Debido a la complejidad de esta herramienta, se procederá a documentar todos los pasos de forma muy detallada, ya que la documentación no lo hace.

5.9 INSTALACIÓN CODEQL

Para empezar con el proceso de instalación de la herramienta, hay que distinguir las dos opciones a seguir. CodeQL ofrece su herramienta como extensión para VSCode y como herramienta CLI (comandos por consola). Las dos opciones permiten realizar las mismas tareas, pero la extensión es más intuitiva y rápida. Por ello, para este proyecto, se ha decidido seguir con extensión.

Para la instalación de la extensión, primero hay que dirigirse a la web de descarga [26] o buscar CodeQL en la sección de extensionesde CodeQL. Una vez instalada la extensión, hay que seguir instalado una serie de dependencias. La primera de ellas es un archivo de dependencias para el correcto funcionamiento de la extensión en el VSCode. Este archivo se descarga desde el repositorio habilitado para ello [27]. Una vez descargado este archivo, se debe instalar en la vista de extensiones de VSCode, en el botón de la esquina superior derecha que muestra tres puntos, es la opción de instalar desde archivo VSIX. A continuación, se muestra una captura como ayuda de este proceso:

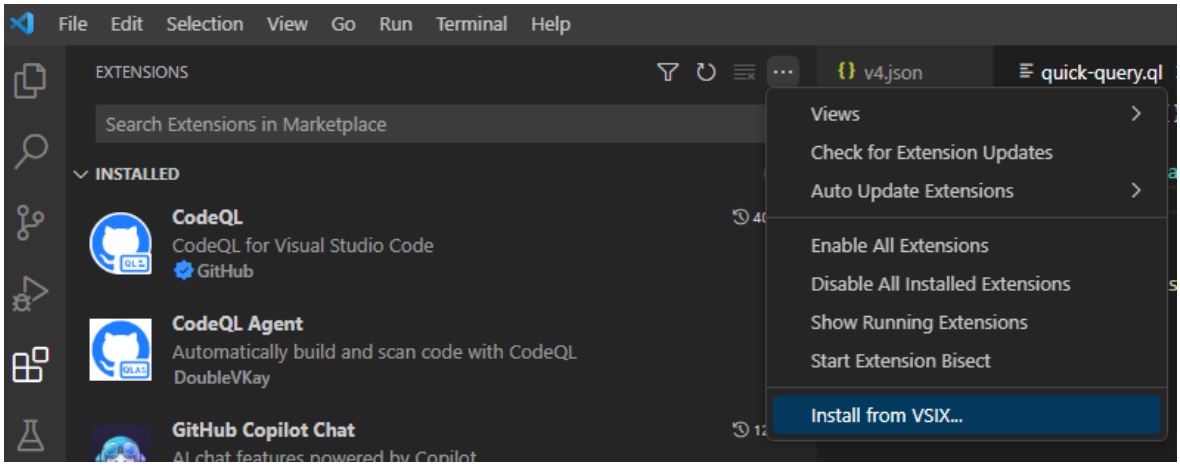


Ilustración 16: Instalación archivo VSIX [27]

Una vez este archivo está correctamente instalado (requiere un reinicio del programa), ya solo quedaría descargarse las últimas dependencias y preparar las bases de datos que se quieren emplear.

Para las dependencias, se necesita añadir al *workspace* el repositorio del código fuente de CodeQL [28]. Esto se debe de hacer ya que, al ejecutar una query, se deben de tener las dependencias correspondientes. Estas dependencias están contenidas en este repositorio y, automáticamente, VSCode las buscará en esta carpeta.

Una vez están las dependencias correctamente importadas en el VSCode, lo último que falta es añadir los repositorios como bases de datos y ejecutar las peticiones. Para importar los proyectos en CodeQL hay varias opciones, la más simple es, desde la extensión de CodeQL en VSCode, en la parte superior derecha, están todos los métodos de cómo importar una base de datos.

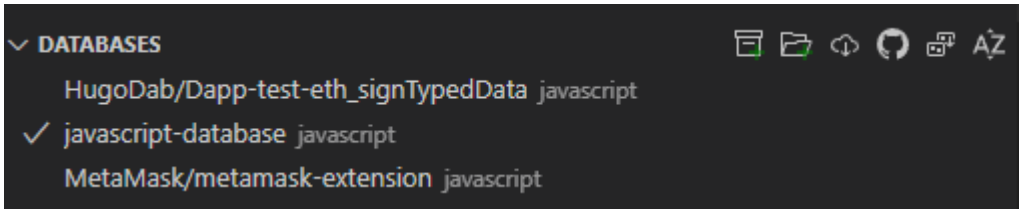


Ilustración 17: Importar una base de datos desde la extensión

La primera opción es importar la base de datos a partir de un archivo específico para ello. La segunda permite importar la base directamente desde un directorio. Las dos últimas opciones son las más rápidas y útiles, la primera de ellas permite importarlo desde un zip en la nube y la última permite importarlo directamente desde GitHub. Esta última opción es la que se suele recomendar, ya que supone un gran ahorro de tiempo y de espacio en el dispositivo.

En muchos casos se encuentran repositorios de código que se quieren analizar y no disponen de una base de datos para CodeQL o dan error al importarlos. Para ello, la solución es crear la base de datos de forma manual y posteriormente importarla en forma de directorio (el segundo método citado anteriormente). Este paso es de vital importancia, porque la gran mayoría de repositorios que se han recolectado en la fase de investigación, no contienen la versión para CodeQL. Para empezar, primero se debe de descargar el repositorio deseado, en este caso lo llamaremos *repo_prueba*. Para crear la base de datos para poder importar en la herramienta, se debe hacer uso del siguiente comando:

```
codeql database create --language=javascript --source-root
  <folder-to-extract> <output-folder>/javascript-database
```

Este comando, para empezar, con la flag de language, hay que establecer el lenguaje de programación del repositorio. Luego, <folder-to-extract> será reemplazado por la ruta completa del repositorio del que se desea crear la base de datos (en este ejemplo es *repo_prueba*). Por último <output-folder> será la ruta donde se encontrará la carpeta /javascript-database que es donde se contendrá la base de datos y la carpeta que se debe seleccionar para importar en VSCode.

Para que este comando funcione, se debe de instalar CodeQL en la versión CLI. Esta es la versión que permite ejecutar comando de CodeQL desde la consola del sistema operativo pertinente. Los pasos a seguir para instalar esta versión son, primero descargar e instalar la herramienta desde la documentación oficial [29]. Para acabar y que este comando funcione correctamente desde cualquier directorio del equipo, se debe añadir la carpeta descargada al Path del sistema operativo. Este último paso es distinto según el sistema operativo, pero fácil de aplicar y es una práctica muy común (fácil de encontrar en internet).

5.9.1 EJECUCIÓN Y PRUEBA DE CODEQL

Una vez instalado el programa, es muy importante validar la instalación ya que, durante el desarrollo de este proyecto, se ha concluido que la instalación falla en muchos casos por errores humanos. Debido a esto, se procederá a verificar dicha instalación.

Para probar que la instalación es correcta, se procederá a realizar una de las pruebas documentadas en la documentación [30]. El primer paso que se debe seguir es añadir el repositorio de pruebas a las bases de datos de CodeQL. Para ello, como se citó en el apartado de instalación, recomendamos importar la base de datos [31], con el botón del logo de Github, ya que es más rápido y sencillo. Este proceso puede tardar unos minutos, ya que el repositorio es de gran tamaño.

Una vez importado el proyecto, se procederá a crear el archivo de peticiones. Para ello, lo más fácil es aprovechar la opción de *quick-query* que ofrece CodeQL. Para ello, se debe ejecutar esta opción desde los comandos de VSCode accediendo con ctrl + shift + P. Una vez abierta la consola, se busca query y aparecerá la opción, que se debe de seleccionar y ejecutar. Esto abrirá un directorio "virtual" dentro del editor de código con todos los archivos necesarios para la ejecución de las queries. El archivo principal es *quick-query.ql* donde se programará la petición que se desea realizar. Para ir verificando el correcto funcionamiento, al ejecutar el comando quick query, en el archivo .ql mencionado anteriormente, debería aparecer el siguiente código por defecto:

```
Import javascript
```

```
Select “”
```

Como se puede observar, este código tiene un aspecto muy similar a SQL, pero hasta aquí llegan las similitudes, ya que cuando se procede a realizar alguna petición avanzada, este código difiere mucho del que se podría encontrar en una petición clásica de SQL. Este código por defecto se reemplazará por el deseado, es este caso y para confirmar el correcto funcionamiento de esta herramienta, ejecutaremos la siguiente petición:

```
Import javascript

From Expr e and
       e.getParent() instanceof ExprStmt
select e, “This expression has no effect.”
```

Por último, solo queda ejecutar dicha petición. Para ello, debemos de seleccionar la base de datos sobre la que queremos ejecutarla (en la pestaña de CodeQL dentro del editor) y seleccionar con el comando ctrl + shift + P, el comando de CodeQL, *run query*. De este comando hay dos versiones, la que permite ejecutar la petición sobre múltiples bases de datos o solo sobre la seleccionada (como se procede a hacer en este paso).

El proceso de ejecución de la petición puede llevar varios minutos (sobre todo si es un repositorio grande o se ejecuta sobre múltiples bases de datos). Después de la compilación, verificación y la ejecución de este, en la sección de la derecha del editor debería de aparecer una ventana con el resultado de la petición, que debería de tener un aspecto similar a lo que se muestra (en caso de que todo se haya realizado de forma exitosa y se podrá verificar el correcto funcionamiento de la herramienta):

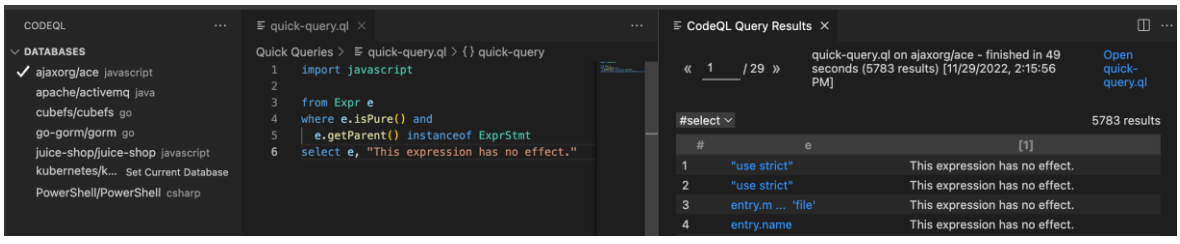


Ilustración 18: Resultado correcto de la ejecución de prueba

Otro archivo muy importante para el funcionamiento de esta herramienta es el *qlpack.yml*. Este es el archivo que se encarga de determinar las dependencias que estarán disponibles para el archivo *quick-query.ql*. Este archivo tiene el siguiente formato, que se comentará posteriormente:

```
Name: vscode/quick-query
Versión: 1.0.0
Dependencies:
       Codeql/javascript-all: '*'
```

- Name: nombre que se le asigna a estas dependencias, este nombre normalmente se suele establecer igual que la ruta donde este contenido este archivo.
- Versión: versión del archivo, por defecto se usa la 1.0.0.
- Dependencias: en esta sección es donde se establece la lista de todas las dependencias que estarán disponible para el archivo de la petición.
    - Codeql/javascript-all: esta entrada de las dependencias carga todos los módulos relacionados con javascript. Es muy importante, como se citó en el apartado de instalación, tener en el workspace de vscode la carpeta main del repositorio de codeql. Esto se debe a que esta dependencia que se establece en este archivo pone la referencia a dicha carpeta donde están contenidos todos los archivos. Por otro lado, la expresión '*' hace referencia a que, de dicha carpeta del main con las dependencias, se desea importar todo el contenido disponible.

La gran mayoría de fallos detectados en esta fase, son provocados por que no se han importado la carpeta del main del código de CodeQL y por la instalación del archivo VSIX.

5.9.2 DETECCIÓN DE VULNERABILIDADES

En esta sección se van a citar todas las vulnerabilidades estudiadas. Para la detección de todas estas vulnerabilidades se ha empleado CodeQL y sus peticiones para encontrar dichos fallos y determinar la seguridad de las aplicaciones.

En esta sección sólo se comentará el proceso de la detección y el código empleado para ello. Posteriormente, se implementará todo el conocimiento que nos proporcionan estas peticiones, con una herramienta, que automatice todo el proceso. El punto diferencial del proyecto es dicha herramienta que, dato un repositorio o una aplicación descentralizada, ofrezca datos sobre la seguridad de esta. De esta forma, el usuario puede determinar antes del uso de la aplicación, su seguridad.

5.9.2.1 Detección eth_sign, personal_signy eth_signTypedData

Como ya se citó en el apartado de métodos de firma, estos dos métodos son inseguros. Esto se debe a que no ofrecen al usuario un texto legible para la firma o que no emplean distintas curvas elípticas según el dominio, para crear distintas claves. En cuanto a *eth_signTypedData*, la única versión que puede llegar a ser segura es la versión 4. Por ello, también se detectarán las versiones 1 y 3 como inseguras. Debido a esto, la propia presencia de estos métodos ya son una vulnerabilidad para el usuario.

Para detectar estos métodos, se procederá a programar una petición de CodeQL que encuentre dichos métodos y nos aporte información sobre la seguridad. Para ello, empleando el tipo Expr (expresión), podemos detectar si hay alguna expresión en el repositorio estudiado que contenga el texto con el nombre de los métodos que se estudian. Para las pruebas, también se ha añadido la condición que permite limitar la búsqueda a un archivo en concreto (en este caso, al que contiene la lógica y las llamadas de dichos métodos). En caso de querer que este estudio se realice en toda la base de datos, esta línea debe de ser comentada o eliminada.

```
/**
 * @name signing_functions.ql
 * @kind expresion has string value
 * @description Find all signing methods and their functions
 * @id javascript/signing_functions
 */
import javascript


from Expr e
where  e.getFile().getAbsolutePath() = "C:/Users/oleob/Downloads/test-dapp-
main/src/index.js" // Replace with your path or delete to search in all dabatase
```

```
and (e.mayHaveStringValue("eth_signTypedData")
or e.mayHaveStringValue("eth_signTypedData_v3")
or e.mayHaveStringValue("eth_sign")
or e.mayHaveStringValue("personal_sign")
)
select e as signing_methodes, e.getEnclosingFunction().getName() as
function_name, e.getEnclosingFunction() as function,
e.getEnclosingFunction().getLocation().getStartLine()
```

Al ser este el primer código ql que se muestra, se explicará en detalle su funcionamiento (posteriormente, no se comentará cada código en detalle, ya que las estructuras y lógicas son las mismas). El código empieza con una serie de metadatos, los cuales son opcionales. Estos metadatos contienen una serie de identificadores, precedidos por el símbolo @. Los identificadores usados son: *id*, especificando el nombre del archivo; *kind* que especifica el tipo de petición que se realiza; *description*, que es una breve descripción en texto del funcionamiento del archivo; *id*, que es un identificador único que permite a ql buscar rápidamente este archivo.

Posteriormente se importa el módulo de javascript, ya que se trata de una base de datos donde la lógica está programada en javascript. Este módulo también se emplea para archivos typescript. El siguiente paso es *from*, donde se crea una variable de tipo Expr (expresión) y se le asigna el nombre e. Posteriormente viene el *where*, que es donde se establecen todas las restricciones que se desean en la petición para que la salida sea la deseada. Dentro de esta sección, se limita el archivo donde se realiza la búsqueda (como ya se ha comentado), y se busca que tenga el contenido deseado. En esta sección es muy útil el uso de operadores lógicos para concatenar restricciones.

Por último, con *select*, se establece toda la información que se muestra al usuario en el formato de salida. En esta sección también se pueden aplicar métodos sobre las variables, ya que con esto se consigue una salida en el formato deseado. Para devolver más de una unidad de información, se separa con comas, siendo cada expresión entre comas, una columna de la salida.

Esta salida, tiene forma de tabla, con tantas columnas como *outputs* se han establecido en la sección de *select*, y tantas filas como resultados que hayan cumplido con las condiciones del *where*. Este código puede ser encontrado en el repositorio de Github creado para este proyecto, en el archivo *signing_functions.ql* [32].

#	signing_methodes	function_name	function	code_line
1	'eth_sign'	onclick	async (... }\n }	1147
2	'personal_sign'	onclick	async (... }\n }	1167
3	'personal_sign'	siweSign	async (... }\n }	1188
4	'eth_signTypedData'	onclick	async (... }\n }	1297
5	'eth_si ... ata_v3'	onclick	async (... }\n }	1364

Ilustración 19: Ejemplo de detección de métodos vulnerables [32]

5.9.2.2 Detección de código no ejecutable

Un fallo muy común a la hora de crear una aplicación es añadir líneas de código que realmente son imposibles de ejecutar. Un ejemplo muy sencillo de esto sería un if con un true y un else. Como el if siempre es cierto, la condición falsa nunca se ejecutaría. Este es un ejemplo muy básico, pero en el mundo real de los programadores, estas cosas ocurren a mayor escala.

Para comprobar esto, se suele realizar un grafo de control de flujo o, más comúnmente conocido como, *control-flow graph* o CFG. Con CodeQL, se puede crear el grafo entero y ver los caminos posibles de ejecución del código. Esto es muy complejo hasta para un ordenador, por lo que solo permite ejecutar estos caminos si son menos de 4 los resultantes. Debido a esto, en vez de conseguir todos los caminos posibles, la herramienta nos proporciona todas las líneas de código que no son alcanzables en las ejecuciones.

Es muy importante hacer hincapié en que, las líneas de código no ejecutables no son ningún tipo de vulnerabilidad en sí, pero esto certifica que, en desarrollo, producción y postproducción, no se han establecido unos mínimos de calidad y auditoría del código. De cara al usuario, todos prefieren usar una aplicación a la que se le han hecho una serie de test, análisis y auditorías tanto de producción como de seguridad.

Debido a esto, es muy probable que, como no han realizado las pruebas pertinentes, haya otros fallos que no se hayan detectado y que suponga una vulnerabilidad. Por ello, si se detectan líneas de código no ejecutables, también se reporta como vulnerabilidad. A continuación, se muestra un ejemplo de una base de datos que contiene código no ejecutable. Este código, al igual que el anterior, busca en un archivo determinado, pero, de cara a realizar el estudio de las aplicaciones, se ejecutará sobre toda la base de datos y, en caso de detectar líneas de código no ejecutable, se le reportará al usuario como vulnerabilidad. El código usado para el ejemplo y desarrollado para el proyecto está contenido en el repositorio de Github como *unreachable_code_file.ql* [33].

#	[0]	[1]
1	return!1	This statement is unreachable.
2	return!1	This statement is unreachable.
3	var n,i,o	This statement is unreachable.
4	var a,s	This statement is unreachable.
5	return!1	This statement is unreachable.
6	return!1	This statement is unreachable.
7	var e	This statement is unreachable.
8	var r	This statement is unreachable.
9	var r	This statement is unreachable.
10	var r,o	This statement is unreachable.
11	var i	This statement is unreachable.
12	var e	This statement is unreachable.
13	var i,a,s	This statement is unreachable.
14	var i,a,s	This statement is unreachable.
15	var e	This statement is unreachable.
16	var e	This statement is unreachable.
17	var e	This statement is unreachable.
18	var u	This statement is unreachable.

Ilustración 20: Ejemplo de detección de código no ejecutable [33].

5.10 PYTHON SCRIPTS

En este apartado se va a comentar los códigos de Python que se han desarrollado para la obtención de los resultados pertinentes. Estos códigos tienen objetivos muy claros y todos, forman parte de la herramienta desarrollada para la detección de vulnerabilidades de las aplicaciones.

5.10.1 FILE SCANNER

El primer script de Python desarrollado es el encargado de estudiar el directorio del código fuente de la aplicación, así como sus archivos y subdirectorios. El funcionamiento de este programa es similar a los vistos en los scripts de sh y en el código de ql que analiza los repositorios en busca de los métodos buscados.

El programa busca dentro del repositorio *eth_signTypedData_v4*, *eth_signTypedData_v3*, *eth_signTypedData*, *eth_sign* y *personal_sign*. En caso de que se encuentre cualquier método que no sea *eth_signTypedData_v4*, se reportará al usuario que la aplicación contiene una vulnerabilidad debido a ello.

Como se ha comentado anteriormente, el único método que *a priori* se puede considerar seguro es el *eth_signTypedData_v4*. El resto de los métodos ya están deprecados, o se conocen fallos de seguridad. Esto no quiere decir que sea certero que son inseguros o maliciosos, pero no es recomendable su uso. En cuanto al método que no se considera inseguro, requiere que posteriormente se realice un estudio en profundidad del método y de su definición de las estructuras de los datos que firman, como se citará a continuación. Este archivo se encuentra en el repositorio de Github de este proyecto, en el archivo *FileScanner.py* [34].

5.10.2 COMPROBACIÓN DEL CONTENIDO

Como se ha citado, para *eth_signTypedData_v4*, aunque sea seguro, se deben de realizar una serie de pruebas para comprobarlo. La primera comprobación que se debe de hacer es que los datos a firmar contienen toda la información mínima para que la firma sea exitosa. Esta estructura mínima es, que contenga un json con las siguientes claves: domain, primaryType, message y Types. Como se citó, esta estructura es válida, aunque esté vacía.

Una vez comprobada que esta estructura es correcta, se debe comprobar el valor asociado al primary type, y todos los usados en el dominio y en el message, están definidos en types. Para empezar, en cuanto al dominio, están establecidas las variables que se pueden emplear: *name, version, chainId, verifyingContract, salt*. También, para estos parámetros, están

establecidos los tipos de datos asociados: *string*, *string*, *uint256*, *address*, *bytes32*. Esto también se comprueba en Python ya que deben de estar los valores dentro de los citados y ninguno añadido. Otra comprobación muy importante en esta fase es detectar si la aplicación, en el dominio, tiene especificado el *verifyingContract*. Esto es importante, porque es el contrato que hay en la red de bloques que verifica esta firma. La forma de que esto sea seguro es gracias a la tecnología propia del blockchain. En caso de que este valor no esté presente, significa que se verifica la firma fuera del blockchain, o conocido como *offchain*, por lo que esto es una puerta abierta a los ataques. En caso de no detectar este contrato, se avisará al usuario.

Por último, se comprueba que todas las variables empleadas en la firma están definidas en la variable de types. Esta es la parte crítica del script y la más compleja. Ya que cada aplicación es muy flexible y esta estructura de datos es muy distinta según la aplicación estudiada.

Para la obtención de estos datos y su análisis, la técnica empleada es muy similar a la conocida como *web scraping*. Esta técnica es muy empleada en páginas web y su objetivo es recopilar datos de las páginas de internet. Se ha basado en esta tecnología, pero adaptada al estudio de código fuente de las mismas. Este código tiene los siguientes pasos:

- Búsqueda de *eth_signTypedData*.
- Búsqueda de JSON.stringlify donde se pasa a formato json la estructura de los datos que se van a firmar.
- Del paso anterior, se obtiene el nombre de la variable que contiene los datos, o si el método anterior se aplica directamente sobre la estructura, se obtiene esta última.
- En caso de obtener el nombre de la variable, luego se procede a obtener la estructura.
- Se da un formato JSON a toda la estructura, ya que, aunque parezca que ya es un json, al estar dentro de un JavaScript, se debe de adaptar al formato json tradicional para trabajar fácilmente con el en Python.
- Se separa de la estructura de datos, el mensaje, el tipo principal, el dominio y los datos.

- Se comprueba que estos fragmentos no son nulos.
- Se procede a comprobar que las variables del dominio están todas definidas dentro de los posibles valores, al igual que los tipos correspondientes. También se comprueba si el dominio tiene un contrato en la red de bloques para confirmar las firmas o no.
- Se extraen la lista de tipos, se obtiene el tipo primario y se comprueba que está definido.
- Se obtiene el mensaje y sus subtipos, ya que lo más común es que el mensaje sea un tipo de dato (el tipo primario) y que tenga otros subtipos definidos dentro de la estructura.
- Se comprueba que las variables del mensaje concuerdan con los tipos y subtipos correspondientes.

Mientras se van ejecutando todos estos pasos y pruebas, se va generando una cadena de texto con todos los problemas y vulnerabilidades detectadas durante el paso de la ejecución. Esta cadena de texto es devuelta por el método, que posteriormente se puede mostrar al usuario para aportar la información de las vulnerabilidades detectadas en la aplicación estudiada.

Los códigos empleados para este estudio y detección de vulnerabilidades son *domainCheck.py* [35] y *signTypedData_checks.py* [36].

5.10.3 APLICACIÓN

En la carpeta de Python Scripts, también está contenido el archivo *window.py* [37]. Este archivo, a su vez llama o contiene métodos de otros archivos para conseguir el objetivo. Se ha empleado la máxima modularidad posible para que el mantenimiento y el desarrollo de la aplicación sea lo más rápido y sencillo posible.

Este archivo principalmente contiene la lógica básica necesaria para gestionar la interfaz gráfica del usuario. Esto quiere decir que gestiona las distintas ventanas, con sus textos, botones y las funcionalidades de estos.

La pantalla principal se compone de una lista despegable con todas las aplicaciones estudiadas en la fase de recopilación de datos y en la creación de la base de datos. Para estudiar la seguridad de una aplicación, se debe de seleccionar dicha aplicación desde la lista despegable y al pulsar el botón, aparecerá una ventana nueva con los resultados del análisis.

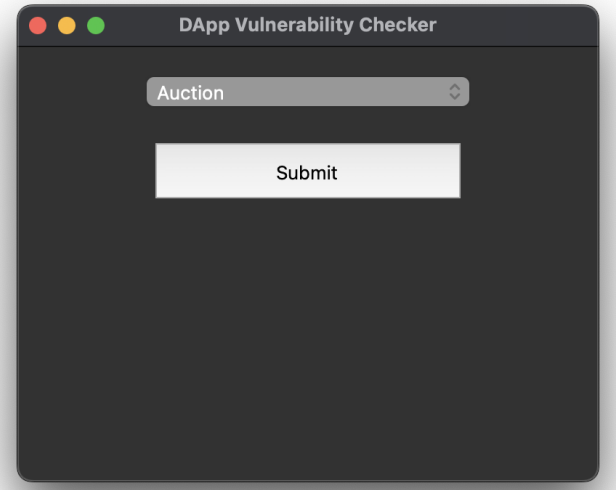


Ilustración 21: Pantalla principal de selección de aplicación

La pantalla con los resultados del análisis contiene una serie de datos obtenidos de la aplicación, como nombre, lenguaje de programación empleado, y la presencia o no de los métodos *eth_signTypedData*, *eth_sign* y *personal_sign*. Esto último, como se ha mencionado en varias secciones, es muy importante para empezar a determinar las vulnerabilidades de la aplicación.

En caso de que la aplicación escogida contenga *eth_signTypedData*, se llevará a cabo un análisis más en profundidad, en el que se estudiará la versión empleada, así como los contenidos del método. El estudio que se realizará en este caso es el citado en la sección

anterior. A continuación, se muestra esta pantalla con los resultados del análisis en caso de estar este método presente o no:

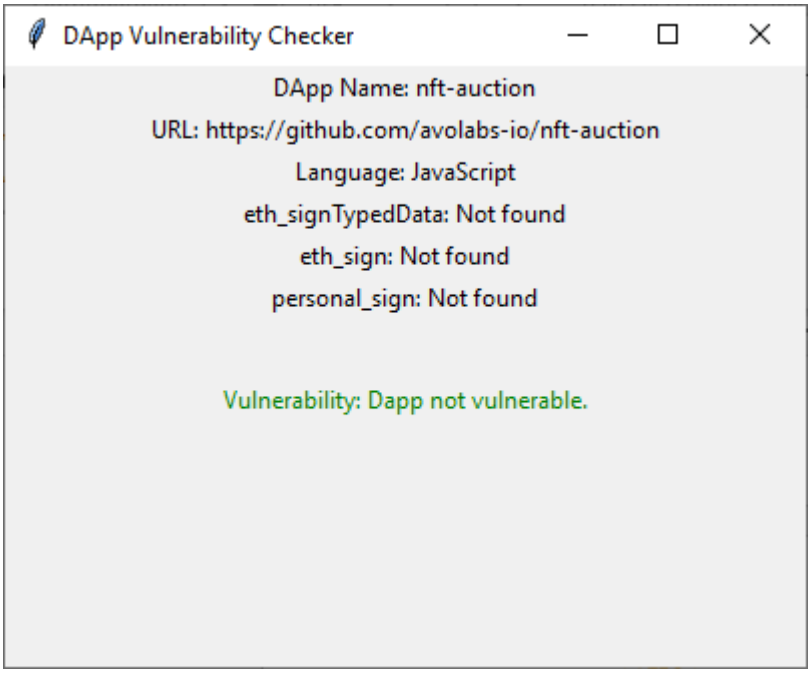


Ilustración 22: Ejemplo de análisis de aplicación no vulnerable

Como se puede apreciar, en la ilustración anterior, en la pantalla de análisis de la aplicación, se muestra el nombre, URL, lenguaje de programación y la presencia de los métodos. Este análisis que se muestra, el resultado del análisis es exitoso, ya que no hay presencia de ninguno de los métodos estudiados (especialmente los que se consideran inseguros). A continuación, se muestra el resultado de dos casos distintos de errores detectados:

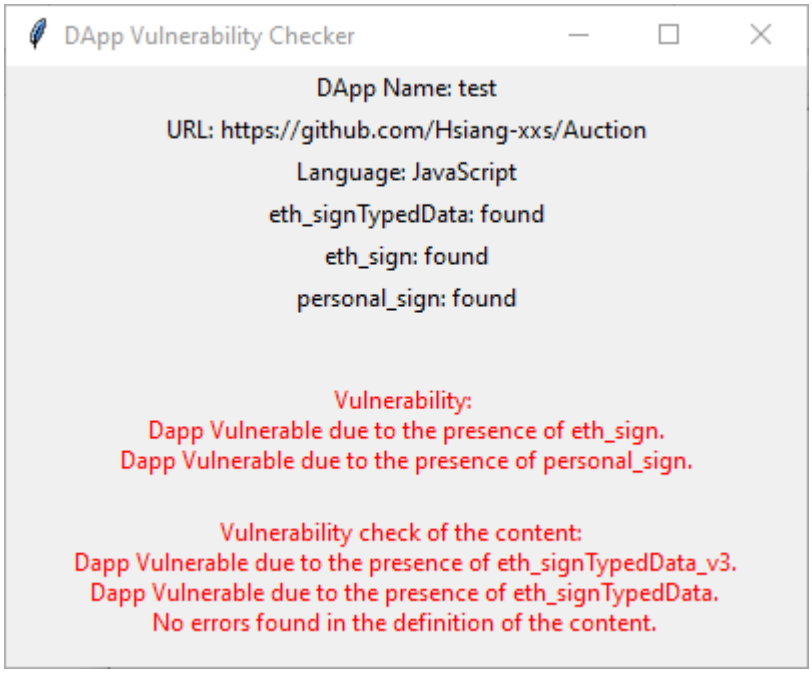


Ilustración 23: Ejemplo de análisis de aplicación vulnerable 1

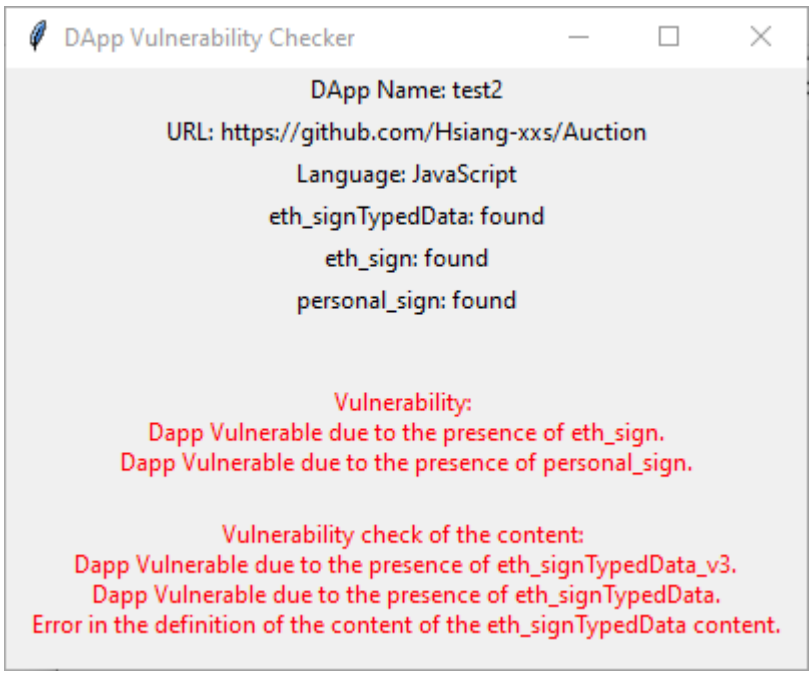


Ilustración 24: Ejemplo de análisis de aplicación vulnerable 2

En la ilustración 23 se muestra cómo en una aplicación de prueba, se ha detectado *personal_sign* y *eth_sign*. En estas dos últimas pruebas, no solo se detectan estos dos métodos, sino que también, en caso de encontrar el método *eth_signTypedData*, se procede a realizar todo el análisis ya comentado sobre sus versiones y el contenido ya citado anteriormente. En el caso de la ilustración 23, en el análisis de este último método, se considera como vulnerabilidad las versiones 1 y 3, por los problemas ya citados. En el análisis del contenido no se ha encontrado ninguna irregularidad.

En cuanto a la ilustración 24, se sigue considerando insegura por las mismas razones que la ilustración anterior, pero, además, en esta aplicación se ha encontrado algún tipo de irregularidad en la definición del contenido que se firma.

Capítulo 6. ANÁLISIS DE RESULTADOS

Con todo lo desarrollado durante el proyecto, se pueden sacar varios resultados claves sobre la seguridad de las aplicaciones descentralizadas. El primero de ellos es que, los métodos de seguridad que se proporcionan no son suficientemente seguros. También queda claro el poco desarrollo y mantenimiento de los distintos mecanismos de seguridad.

Queda muy claro que, la presencia de varios de estos métodos estudiados ya supone una vulnerabilidad. Como se ha citado varias veces, si estos métodos se implementan de forma correcta, no tiene por qué haber un riesgo, pero es más propenso a ello. Como recordatorio, el método *eth_sign* tiene como principal problema que, usa la misma curva de cifrado para distintos dominios. Esto es un gran problema ya que, es mucho más fácil conseguir las claves cuando se usa una misma clave para muchos ámbitos distintos. El otro problema principal es que, el mensaje firmado que se presenta al usuario no es legible. Debido a esto, se puede generar una firma falsa con un contenido malicioso y el usuario no tiene herramientas disponibles para confirmar el contenido de este. Por último, este método está dentro de la categoría de los métodos deprecados. Estos métodos son los que no se eliminan de las aplicaciones, es decir, se pueden seguir usando, pero no tienen soporte ni mantenimiento, por lo que no se recomienda su uso.

En cuanto a *personal_sign*, soluciona que la firma no sea legible, pero es un método que es muy complejo. Esto hace que el método no sea eficiente. Por otro lado, tiene problemas de compatibilidad con varias carteras de criptomonedas, por ello, es un método poco empleado y utilizado.

Por último, el método *eth_signTypedData* es el último incorporado y el más seguro. Como ya se ha citado múltiples veces, este es el método que más se recomienda implementar en las aplicaciones, debido a su seguridad y, sobre todo, a su gran flexibilidad. El principal cambio que incorpora este método es que define un dominio en el que puede ser empleado, acabando con el problema de usar la misma curva de firma para distintos dominios de

seguridad. En este método hay varias versiones (1, 3 y 4), siendo solo la versión 4 la recomendada, debido a que las anteriores no tenían definido el separador de dominio y por ello, son propensas a un *replay attack*. A pesar de todo esto, que sea un método muy flexible, añade una complejidad añadida y por ello, problemas de seguridad. Este método no comprueba el contenido que se firma, por lo que puede estar mal definido y la firma será exitosa.

Debido a todo lo anterior, se ha desarrollado la aplicación que muestra toda la información sobre la seguridad de las aplicaciones estudiadas. Gracias a esto, no recae al completo, la responsabilidad sobre el usuario, de estudiar la seguridad de la aplicación. La aplicación es una herramienta que ayuda a esto y proporciona información útil previa al uso de una aplicación.

Gracias a la aplicación a la base de datos generada durante el proceso de investigación, se procede a realizar un análisis detallado sobre las aplicaciones recopiladas. Con lo mencionado y con las conclusiones obtenidas sobre que se considera una vulnerabilidad, se pueden sacar unas estadísticas representativas sobre estas aplicaciones.

A continuación, se muestran unas gráficas obtenidas a partir de un breve análisis sobre la base de datos obtenida de las aplicaciones. Para obtener estas gráficas, se ha realizado un breve estudio de la presencia de los distintos métodos vulnerables y de posibles métodos vulnerables. Esta base de datos está compuesta de 771 aplicaciones, lo cual no da un resultado algo representativo sobre todo el “mercado” de aplicaciones distribuidas. El resultado de estas gráficas se analiza en el Capítulo 7:

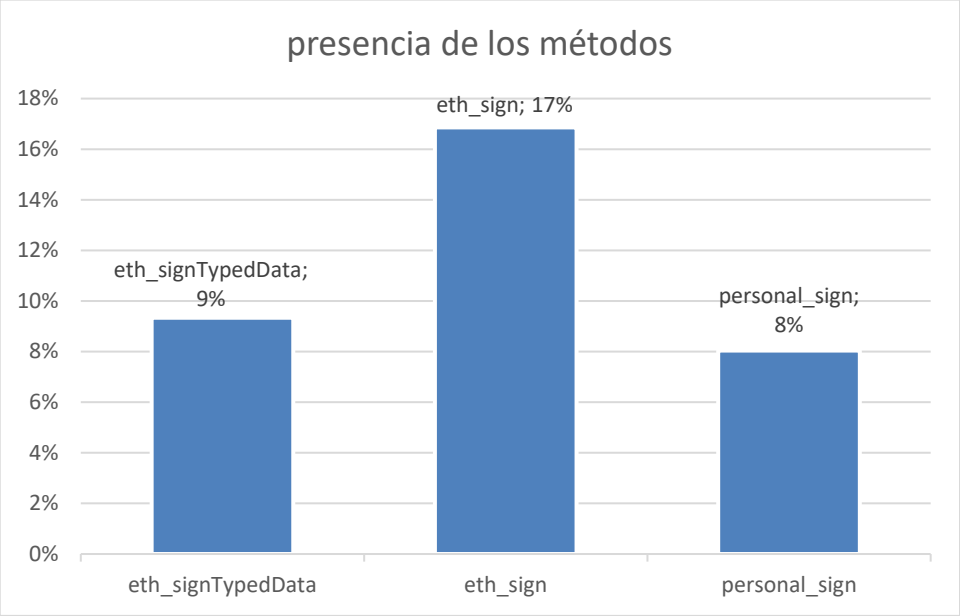


Ilustración 25: Estadística de la presencia de los métodos estudiados

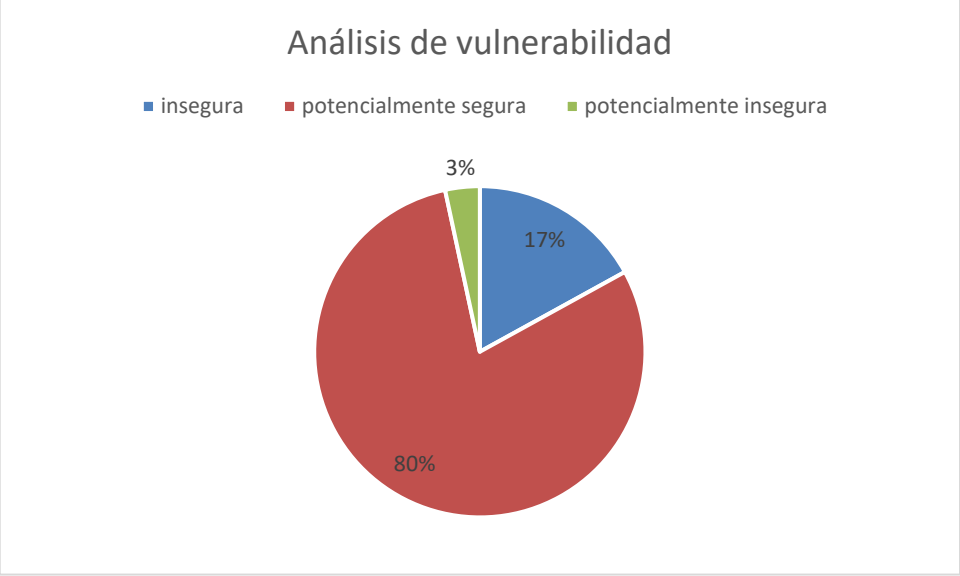


Ilustración 26: Estadística sobre la presencia de vulnerabilidades

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Gracias a todo el proceso de investigación realizado en este trabajo, la base de datos de las aplicaciones estudiadas y de la herramienta creada, se pueden analizar los resultados mostrados en el Capítulo 6.

Para la obtención de las gráficas de dicho capítulo, se ha realizado un estudio sobre la base de datos creada para el estudio de las aplicaciones. Para la obtención de los resultados de la ilustración 25 (que más adelante se comentará), se ha estudiado, de todas las aplicaciones de la base de datos, la presencia de los tres métodos objetivo de este estudio. Gracias a las distintas herramientas (VScode, CodeQL y Python) se ha llegado a determinar qué aplicaciones contienen los distintos métodos. De esta forma, se ha logrado obtener la gráfica de la ilustración 25. Esta ilustración muestra los porcentajes de aparición de los distintos métodos. Como se puede apreciar, el método *eth_signTypedData* aparece en un 9% de las aplicaciones, *eth_sign* es el que más aparece con un 17% de las aplicaciones y *personal_sign* con un 8%.

En cuanto a la ilustración 26, su información es derivada de la gráfica anterior. En esta gráfica se muestra el porcentaje de aplicaciones de la base de datos que se consideran vulnerables (17%), las que son potencialmente inseguras (3%) y las que se pueden sonsiderar seguras 80%). Para determinar las aplicaciones que se consideran inseguras, se ha determinado gracias al estudio realizado en este trabajo, ya que los métodos *eth_sign* y *personal_sign* son consideramos vulnerables (ver capítulos anteriores para la explicación). Debido a esto, cualquier aplicación que contenga algunos de estos métodos, o los dos, es considerada vulnerable. De la misma forma, las aplicaciones que contengan *eth_signTypedData* son consideradas potencialmente vulnerables ya que, como se ha desarrollado en este trabajo, depende de la versión y de la definición del contenido, se puede incluso a llegar a determinar que la aplicación es vulnerable. El porcentaje de las aplicaciones que se consideran potencialmente vulnerables debida a esta razón es del 3%, lo que es un porcentaje bastante reducido. Por otro lado, si tenemos en cuenta cualquier

aplicación que contenga este método, el porcentaje asciende al 9%, ya que muchas aplicaciones que implementan dicho método también implementan los métodos considerados vulnerables.

Por último, el resto de las aplicaciones, a priori, no se consideran vulnerables. Este grupo son el 80% de las aplicaciones estudiadas. Este grupo es amplio, ya que no todas las aplicaciones estudiadas están programadas en TypeScript o JavaScript. Debido a esto, los nombres de los métodos no serán los mismos. Por otro lado, muchas aplicaciones no trabajan con Ethereum, por lo que estos métodos tampoco existen.

A pesar de esto último, el 30% de las aplicaciones estudiadas son vulnerables o potencialmente vulnerables. Este es un porcentaje muy alto, contando con que la muestra empleada es grande y representativa del mercado. Por ello, se concluye que, hay un déficit de seguridad muy grande en las aplicaciones que trabajan sobre el blockchain. Esto es un problema muy serio, debido a las grandes cantidades de dinero que se manejan en estas aplicaciones. Esto es comparable a que los sistemas de los bancos, así como sus aplicaciones móviles y web, tuviesen muchos problemas de seguridad.

Es muy importante la herramienta desarrollada en este proyecto, ya que los usuarios pueden analizar las aplicaciones y determinar su seguridad. Gracias a esto el número de estafas se podría reducir drásticamente. Como se ha citado varias veces, el objetivo final es reducir el número de vulnerabilidades, estafas y, por lo tanto, mejorar la experiencia del usuario y la usabilidad de estas aplicaciones. Estas aplicaciones, por parte del backend, son seguras ya que la tecnología de la cadena de bloques es muy segura, pero los desarrolladores, al añadir las capas superiores cometen errores, o simplemente, crean aplicaciones maliciosas.

7.1.1 TRABAJOS FUTUROS

De cara al futuro, se debería de potenciar la lógica y funcionabilidad de la aplicación creada en este proyecto. Un ejemplo de esto sería implementar toda la lógica que determina la presencia de los métodos y, sobre todo, la comprobación del contenido fuente de las firmas, en CodeQL.

Esta herramienta es muy potente y enfocada a ello. En este trabajo no se ha logrado a realizar, debido a la complejidad que esto conlleva. Debido a ello, se ha tenido que contactar con los desarrolladores de CodeQL (Github), para ayudar a este objetivo.

La principal implementación de cara al futuro de este proyecto es la comprobación del contenido de las firmas de *eth_signTypedData*. Con CodeQl, se puede realizar unas queries que repliquen el *webscraping* realizado en Python, pero de forma mucho más eficiente. También se lograría generalizar mucho más esta petición, de forma que funcione correctamente con cualquier aplicación estudiada.

Otra mejora de cara a futuro que no se ha podido realizar por tiempos y complejidad, es la automatización de los procesos de la aplicación. Con esto, se quiere reemplazar la lógica que obtiene los datos de las aplicaciones y sus análisis. Esta fase es relativamente fácil de obtener, ya que todos los scripts necesarios ya han sido creados durante el desarrollo de esta aplicación. Estos scripts son los que obtienen las aplicaciones descentralizadas y sus códigos fuente desde Github, analizan su contenido en busca de los métodos y realizan los resultados. Actualmente la aplicación obtiene las aplicaciones, la presencia de los métodos y sus estudios de la base de datos creada en el proyecto. Con este cambio, se automatiza todo el proceso y se logra generalizar todo el proceso de determinación de las vulnerabilidades de las aplicaciones. Con esto, se lograría analizar cualquier aplicación existente en el internet simplemente con su nombre y se consigue una herramienta completa y útil para desarrolladores y usuarios.

Por otro lado, otro trabajo de cara a futuro sería mantener toda esta lógica de la herramienta desarrollada. Esto ocurre en cualquier software creado, pero concretamente en este es muy importante. Esto se debe a que Metamask puede implementar o eliminar métodos relacionados con las firmas y la seguridad de las aplicaciones y por ello, habría que adaptar la aplicación a estos cambios. Cuando se elimina un método, ya sea seguro o vulnerable, se debe de tratar con especial cuidado. Cuando se añade un nuevo método, no solo se debe implementar en la herramienta, sino que también se debe de realizar de nuevo la fase de investigación de dicho método para determinar su seguridad y como debe de ser analizado.

Por último, en cuanto al *paper* científico, se presentará cuando el proyecto se acaba por parte de Chicago. Este proyecto es la tesis doctoral de un alumno del Instituto Tecnológico de Chicago (IIT). Debido a esto, el *paper* será presentado cuando se presente la tesis.

Capítulo 8. BIBLIOGRAFÍA

[1] Binance. (s. f.). *Binance – Exchange de criptomonedas para bitcoin, Ethereum y altcoins*. https://www.binance.com/es

[2] Sánchez, Á. (2022, 26 abril). Las criptomonedas movieron unos 60.000 millones de euros en España el año pasado. *El País*. https://elpais.com/economia/2022-04-26/las-criptomonedas-movieron-unos-60000-millones-de-euros-en-espana-el-ano-pasado.html

[3] Vanguardia, L. (2022, 10 agosto). Un hacker roba un NFT por valor de 310.000 euros y lo acaba vendiendo por casi la mitad. *La Vanguardia*. https://www.lavanguardia.com/tecnologia/20220810/8459460/roban-nft-valor-310-000-euros-hacker-acaba-vendiendo-mitad-pmv.html

[4] The crypto wallet for Defi, Web3 Dapps and NFTs | MetaMask. (s. f.-b). https://metamask.io/

[5] *CodeQL*. (s. f.). https://codeql.github.com/

[6] *GitHub: Let's build from here*. (s. f.-b). GitHub. https://github.com/

[7] *Node.js*. (s. f.-b). Node.js. https://nodejs.org/es

[8] Echarri, M., Echarri, M., & Echarri, M. (2022, 27 julio). El chico de 15 años que robó 24 millones en criptomonedas : Ellis Pinsky cuenta su historia pero nadie sabe si creerle. *El País*. https://elpais.com/icon/2022-07-27/el-chico-de-15-anos-que-robo-24-millones-en-criptomonedas-ellis-pinsky-cuenta-su-historia-pero-nadie-sabe-si-creerle.html

[9] DappRadar - The World's Dapp Store | Blockchain Dapps Ranked. (s. f.). DappRadar. https://dappradar.com/

[10] web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation. (s. f.). https://web3js.readthedocs.io/en/v1.10.0/

[11] *Ether.js Documentation*. (s. f.). https://docs.ethers.org/v5/

[12] *Home | Uniswap Protocol*. (s. f.). Uniswap Protocol. https://uniswap.org/

[13] *Uniswap Labs*. Código Fuente de Uniswap (s. f.). GitHub. https://github.com/Uniswap

[14] Home | MetaMask developer documentation. (s. f.). https://docs.metamask.io/

[15] *Metamask Playground* (De Metamask [Metamask]). (s. f.). Playground Metamask. https://metamask.github.io/api-playground/api-documentation/

[16] Proposals, E. I. (2019). EIP-2255: Wallet Permissions System. *Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-2255

[17] *MetaMask*. Repositorio de código de Metamask (s. f.). GitHub. https://github.com/MetaMask

[18] Proposals, E. I. (2020). EIP-3085: Wallet Add Ethereum Chain RPC Method (`wallet_addEthereumChain`) [DRAFT]. *Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-3085

[19] *Sign data | MetaMask developer documentation*. (s. f.). https://docs.metamask.io/wallet/how-to/sign-data/

[20] *What is «eth_sign» and why is it a risk?* (2023, 12 mayo). MetaMask. https://support.metamask.io/hc/en-us/articles/14764161421467-What-is-eth-sign-and-why-is-it-a-risk-

[21] Proposals, E. I. (2017). EIP-712: Typed structured data hashing and signing. *Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-712

[22] Jie, K. W. (2020, 20 enero). EIP712 is here: What to expect and how to use it - MetaMask - Medium. *Medium*. https://medium.com/metamask/eip712-is-coming-what-to-expect-and-how-to-use-it-bb92fd1a7a26

[23] Miguelob. (s. f.). GitHub - miguelob/TFM: Repository containing my final masters research project. GitHub. https://github.com/miguelob/TFM

[24] *Installation instructions for GitHub dependencies*. (s. f.). GitHub CLI. https://cli.github.com/manual/installation

[25] Miguelob. (n.d.). *GitHub - miguelob/Dapp-test-eth_signTypedData*. GitHub. https://github.com/miguelob/Dapp-test-eth_signTypedData

[26] *CodeQL*. (n.d.-b). https://codeql.github.com/

[27] Github VSIX download repository. (n.d.). *Releases · github/vscode-codeql*. GitHub. https://github.com/github/vscode-codeql/releases

[28] Github. (n.d.-a). GitHub - github/codeql: CodeQL: the libraries and queries that power security researchers around the world, as well as code scanning in GitHub Advanced Security. GitHub. https://github.com/github/codeql

[29] *Getting started with the CodeQL CLI - GitHub Docs*. (n.d.). GitHub Docs. https://docs.github.com/en/code-security/codeql-cli/using-the-codeql-cli/getting-started-with-the-codeql-cli

[30] *Basic query for JavaScript code — CodeQL*. (n.d.). https://codeql.github.com/docs/codeql-language-guides/basic-query-for-javascript-code/

[31] Ajaxorg. (n.d.). *GitHub Rpositorio de pruebas para validar el CodeQL - ajaxorg/ace: Ace (Ajax.org Cloud9 Editor)*. GitHub. https://github.com/ajaxorg/ace

[32] Miguelob. (n.d.-e). *TFM/Code/queries/signing_functions.ql at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/queries/signing_functions.ql

[33] Miguelob. (n.d.-f). *TFM/Code/queries/unreachable_code_file.ql at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/queries/unreachable_code_file.ql

[34] Miguelob. (n.d.-e). *TFM/Code/python scripts/FileScanner.py at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/python%20scripts/FileScanner.py

[35] Miguelob. (n.d.-e). *TFM/Code/python scripts/domainCheck.py at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/python%20scripts/domainCheck.py

[36] Miguelob. (n.d.-g). *TFM/Code/python scripts/signTypedData_checks.py at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/python%20scripts/signTypedData_checks.py

[37] Miguelob. (s. f.). *TFM/Code/python scripts/window.py at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/python%20scripts/window.py

[38] *Busqueda de Github para Data Mining*. (n.d.). GitHub. https://github.com/search?l=JavaScript&q=ethereum+dapp&type=Repositories

[39] Miguelob. (n.d.-c). *TFM/Code/Data Mining/searchRepo.sh at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/Data%20Mining/searchRepo.sh

[40] Miguelob. (n.d.-b). *TFM/Code/Data Mining/finderGithub.sh at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/Data%20Mining/finderGithub.sh

[41] Miguelob. (n.d.-c). *TFM/Code/Data Mining/mergeFile.sh at main · miguelob/TFM*. GitHub. https://github.com/miguelob/TFM/blob/main/Code/Data%20Mining/mergeFile.sh

ANEXO I: MÉTODOS DEPRECADOS

Como en todas las herramientas, especialmente APIs, hay una categoría denominada métodos deprecados. Esta serie de métodos se refieren a aquellos que, por algún motivo, normalmente de seguridad, se ha descontinuado su desarrollo, pero más importante, su mantenimiento. Esto último es importante, ya que la gran mayoría de problemas de seguridad aparecen cuando se dejan de mantener los equipos o el software.

Uno de los aspectos claves de estos métodos deprecados es que, se suelen categorizar muchas veces así debido a que muchas plataformas todavía las usan. Este es el caso de varios métodos en Metamask, como puede ser *eth_sign*. Este se considera deprecado ya que no se mantiene este método, pero un gran número de aplicaciones descentralizadas todavía hacen uso de este método, incrementando la inseguridad de esta. En el caso de Metamask, se proporciona muy poca información del porqué estos métodos están deprecados. En el caso de *eth_sign* se ha llegado a determinar, después de una profunda investigación, que se debe a que usa la misma curva de cifrado para todos los métodos criptográficos. Esto conlleva un incremento de la posibilidad de que la clave privada sea descubierta.

Metamask contiene una serie de métodos deprecados, los cuales se han documentado también y que se muestran a continuación:

	API name	Brief description	Reason of deprecation	Use instead	Link
2	ethereum.chainId	A hexadecimal string representing the current chain ID	non-standard	ethereum.request({ method: 'eth_chainId' })	Doc
3	ethereum.networkVersion	A decimal string representing the current blockchain's network ID	You should always prefer the chain ID over the network ID	ethereum.request({ method: 'net_version' })	Doc
4	ethereum.selectedAddress	Returns a hexadecimal string representing the user's "currently selected" address	The value of this property can change at any time	ethereum.request({ method: 'eth_accounts' })	Doc
5	ethereum.enable()	Alias for ethereum.request({ method: 'eth_requestAccounts' })	-	ethereum.request({ method: 'eth_requestAccounts' })	Doc
6	ethereum.sendAsync()	This is the ancestor of ethereum.request	It only works for JSON-RPC methods, and takes a JSON-RPC request payload object and an error-first callback function as its arguments	ethereum.request()	Doc
7	ethereum.send()	. It is essentially an overloaded version of ethereum.sendAsync()	This method behaves unpredictably and should be avoided at all costs	ethereum.request()	Doc

Ilustración 27: Métodos deprecados de Metamask

ANEXO II: DATA MINING

En este anexo se contiene todo lo referente a la recopilación de datos masiva para la obtención de resultados. Durante el proyecto ya se ha citado como con un script se ha recopilado datos de ciertos repositorios de Dapps y se analizaba la presencia de los distintos métodos de firma y sus líneas. Esta forma de obtención de los datos es mucho mejor que la manual, pero sigue siendo insuficiente para tener un resultado representativo.

Gracias a este método de extracción masiva y a partir del buscador de Github, se pueden obtener miles de repositorios. Primero, desde el buscador de Github, se puede encontrar todos los repositorios relacionados con DApps y Ethereum y filtrar por el lenguaje JavaScript [38]. Una vez está la búsqueda de repositorios se ha realizado, se procederá al análisis de estos.

Para ello, se han creado una serie de scripts de tipo Shell en los que, utilizando la API de Github, se obtiene todo lo deseado. El primer script es *searchRepo.sh* [39]. Este archivo es el que contiene toda la lógica que involucra la API de Github y la obtención de los repositorios de la lista deseada con los filtros aplicados. Una vez esta lista de repositorios han sido obtenidos de la lista, se procede a ejecutar el script *finderGithu.sh* [40]. Este archivo es el encargado de encontrar, dentro de dichos repositorios, la presencia de una cadena de caracteres. En este caso, se aplicará para buscar los métodos de firma ya citados durante anteriormente en el apartado de firmas. Por último, es muy importante que estos datos no estén sesgados y, para ello, se realiza una limpieza de estos para que no se contengan duplicados de los repositorios analizados manualmente. Toda esta lógica está contenida en el script *mergeFile.sh* [41].

ANEXO III: ALINEACIÓN CON LOS ODS

Este proyecto se alinea principalmente con los objetivos de desarrollo sostenible números 9, 11 y 12. Estos tres objetivos son correspondientemente: industria, innovación e infraestructura; ciudades y comunidades sostenibles; producción y consumo responsables.

En cuanto al primer objetivo y segundo objetivo, blockchain nos ofrece una nueva infraestructura muy útil de cara al futuro, ya que es una red descentralizada y nos permite trazar todos los elementos que se mueven en esta red. Esta infraestructura ya implementa novedades como puede ser el dólar digital, el cual afecta a comunidades. Para que esto sea sostenible, hay que tener en cuenta el punto número 12 de las ODS. Este es un factor clave, ya que se ha comprobado que esta red de bloques es muy poco eficiente. Esto se debe a la gran cantidad de cálculos que se realizan para que esta red funcione de forma segura y descentralizada. Debido a esto, la red necesita una cantidad de energía superior a la que consumen algunos países. Uno de los puntos clave para que esta tecnología prospere es reducir este consumo. Esto ya se implementa en algunas de sus utilidades como puede ser Ethereum, el cual con la versión dos ha cambiado la forma de comprobar transacciones por una que requiere menos esfuerzo computacional y, por tanto, menos consumo.

Debido a todo lo comentado, este proyecto se alinea ciertamente con varios puntos de los presentados por los ODS. Aun así, es una tecnología muy moderna y la cual tiene que superar algunas barreras, como la comentada de la eficiencia energética. Una vez esto se consiga, va a suponer un cambio tecnológico y social muy importante que permitirá a la sociedad desarrollar nuevas tecnologías y el desarrollo económico de forma sostenible y eficiente.