CS597 - Research project - January to August 2023

Vulnerabilities in Decentralized Applications

Hugo DABADIE

Illinois Institute of Technology

10 West 35th Street
Chicago, IL, USA

Internship advisor
Yue DUAN
School advisor
Sylvain BOUVERET

September, 4th 2023

Contents

List of Figures

List of Tables

Glossary

Blockchain Distributed database that maintains a continuously growing list of ordered records, called blocks. 11

ecrecover Function used to retreive the address used in a signature given the unsigned message. 27, 30

Ethereum Ethereum is the community-run technology powering the cryptocurrency ether and thousands of decentralized applications. 12, 28

Replay Attack Form of network attack in which valid data transmission is maliciously or fraudulently repeated or delayed. 15, 20, 27, 28, 29

Smart Contract Computer program that is stored and runs on a decentralized trustless network, such as a blockchain. 3, 15, 27, 28, 31

Acronyms

API Application Programming Interface. 3, 12, 13, 14, 15, 18, 20, 24, 25, 37

DApp Decentralized Application. 3, 8, 9, 10, 11, 12, 15, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34, 37, 38

EIP Ethereum Improvment Proposals. 13, 14, 15

NFT Non-Fungible Token. 15, 16

Acknowledgments

I would like to express my sincere gratitude to the individuals who contributed to the successful completion of this report.

First and foremost, I extend my deepest appreciation to Yue DUAN for his invaluable guidance, expertise, and mentorship throughout the research process. His insightful feedback and encouragement greatly enriched the quality of this work.

I also extend my thanks to Sajad MEISAMI for his precious insights and contributions, which played a significant role in shaping the direction and depth of the study. His expertise and dedication have been instrumental in driving our research forward.

Furthermore, my appreciation extends to Miguel OLEO BLANCO for his diligent efforts, insightful discussions, and collaborative spirit. His contributions added a unique perspective to the project and contributed to its overall excellence.

I am genuinely grateful for the support and guidance I received from every team member, without whom this work would not have been possible.

I. Introduction

At the end of the engineer formation, the student, in France, has to do a 6-month internship called *Projet de Fin d'Etudes* what stands for *End of studies project*. Usually, an engineering student has already done another internship before this one. Ensimag encourages students to do at least one internship in the industry.

The *Projet de Fin d'Etudes* brings the university curriculum to a close, and is an opportunity for students to put into practice the skills acquired during their formation. It also develops their ability to work as part of a team, and prepares them to enter the professional world.

I already have done two 3-month internships in the industry, and this is one of the reasons why I wanted to do this last internship in a research laboratory. I was drawn to cybersecurity, and I wanted to learn more about this field. This project applies cybersecurity to blockchain technology, which is a domain I had not studied during my formation.

I was curious about discovering a new scope of computer science, especially because blockchain technologies are on-trend. Indeed, their decentralized nature, enhancing security, transparency, and efficiency in transactions makes them attractive.

They enable smart contracts, tokenization, and innovations like decentralized finance and supply chain management. Despite challenges, blockchain's potential for disruption, investment opportunities, and real-world applications are driving its popularity.

II. Context

A) Illinois Institute of Technology

Based in the global metropolis of Chicago, Illinois Tech is the only technology-focused university in the city. It offers degrees in engineering, science, architecture, business, design, human sciences, applied technology, and law.

There are around 7,000 students at IIT of whom 3,000 are graduate students. 62% of the graduate students are international coming predominantly from India, China and Spain.

There are several Master proposed by IIT. The Master of Computer Science degree program requires a core curriculum of three courses and seven other elective courses. One of these elective courses can be CS597: Research project.

B) CS597: Research project

This course counts as an elective course at IIT. The student is under the supervision of an IIT professor who gives him a project to work on.

The goal is to be integrated in a research team, working for a project in a field that puts the student formation into practice. The length is variable and depends on the student and on the project.

C) Team

The team is under the supervision of Professor Yue DUAN.

He is an assistant professor of computer science at Singapore Management University since this summer. Before this, he was a Gladwin Development Chair Assistant Professor at Illinois Institute of Technology. He had the postdoctoral training at Cornell University and University of Utah, and obtained his PhD in Computer Science from UC Riverside in 2019.

His research interests mainly in different aspects of Computer Security, including mobile security, system security, software testing, blockchain security and AI security.

Sajad MEISAMI is a PhD student at Illinois Institute of Technology since 2021, and he is responsible for this project. He has obtained a Master of Science from the Sharif University of Technology in 2020.

The team is completed by two Master students. Miguel OLEO BLANCO who does a dual-degree Master at Illinois Institute of Technology, and with Universidad Pontificia Comillas. I am the last student of the team pursuing my dual-degree at Illinois Institute of Technology.

III. Motivations

Since the progress of Blockchain technology, a multitude of security challenges has arisen. These challenges commonly revolve around issues such as phishing attacks, scams, and flaws in the implementation of decentralized apps that operate on the blockchain. As a result, there is a growing necessity for users to ensure the security of these potentially harmful apps. The purpose of this project is to develop a tool capable of assessing the safety of apps by analyzing their source code, thereby aiding end users in making informed decisions.

To accomplish this objective, a comprehensive investigation and research effort will be undertaken, focusing on the security landscape of decentralized applications and blockchain wallets. This will involve an exploration of the methods used in real-world DApps and how they are integrated. Ultimately, the project aims to design and create a tool that can effectively carry out this analysis.

Another motivation of this project is to reduce the scams and thefts that occur in DApps, and in which millions of dollars are wasted every year for this reason. A clear example is one of the most recent thefts, in which 24 million euros in cryptocurrencies were instantly stolen. The goal is to develop a tool that can make the user more aware about the security of the applications that they are using.

IV. Methodology

A) Main objectives

The main purpose of the project is to give the end user of blockchain applications knowledge about the security of the application that he uses. To this end, one objective is the creation of a tool that will be able to determine the security level of the application and present it to the user.

Another objective of this project is to generate a study on the security of the DApps market with the information gathered from analyzing a large number of applications. This study will be presented in paper format to the scientific community after the defense of this work.

Due to the research focus of this project, there are a number of objectives that have been generated while working towards the main objective.

B) Organization

1. Spring semester

The project has begun with the spring semester, starting in early January and ending in May. During this period, I had three other courses, hence, I was not working full-time on the project. This organization is due to constraints with the dual-degree Master that I was pursuing. Indeed, I had to match the requirement from both IIT and Ensimag.

Considering that particular situation, the work was given every week during a meeting. It was done during the week with the help from the other people on the project when needed.

2. Summer semester

Starting from May to August, during the summer semester, I was working full-time on the project. Daily meetings have been set up. The goal was to have a moment to exchange on the issues and the achievement made every day.

However, the project was essentially done online because it was more convenient to meet. Indeed, during the project, there have been some familial issues and also some visa issues that prevent the team from meeting physically. Those unforeseen events have not affected the progress of the project.

V. Development

A) Decentralized Applications

1. Top 100 DApps

The work has begun with finding the most popular DApps. We used the website Dapp Radar [?]. It provides a list of different DApps that can be sorted to get the most visited ones. We chose to sort the DApps according to their traffic.

It is important to get this list because they are more sensible as more people uses them. Moreover, some less popular DApps try to impersonate the style of the most famous ones. Therefore, understanding the issues of the biggest ones can tell about the smallest.

When we started the analysis, the most famous DApps was Uniswap [?]. The V3 and the V2 were in the top 10 DApps (rank 1 and 8). Uniswap defines itself as an automated market maker. In practical terms, it is a collection of smart contracts that define a standard way to create liquidity pools, provide liquidity, and swap assets. The Uniswap Protocol is an open-source protocol for providing liquidity and trading tokens on Ethereum. It eliminates trusted intermediaries and unnecessary forms of rent extraction, allowing for safe, accessible, and efficient exchange activity. The protocol is non-upgradable and designed to be censorship resistant. The Uniswap Protocol and the Uniswap Interface were developed by Uniswap Labs.

2. Other DApps

Many more DApps exists. One way to find a lot more of them is to make a search on GitHub. Indeed, the majority of DApps are Open-source and their source code are available on this website. Using this method, we can easily find more than 58,000 GitHub repositories containing the keyword *dapp* as the figure 1 illustrates.

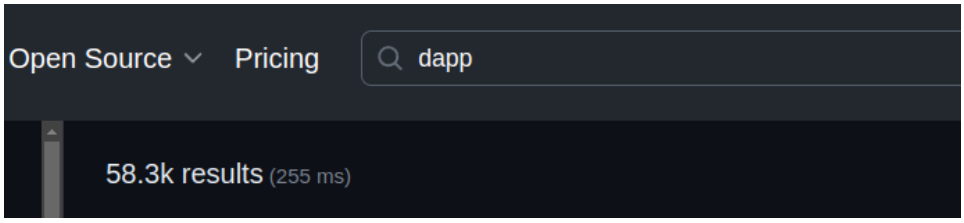

Figure 1: GitHub results for dapp search

Finding more DApps is essential for two main reasons. The first one is that if we only analyze the top 100 DApps, we will miss a very high percentage of the existing DApps.

The other one is that usually, the more something is used, the more secure. Therefore, finding a vulnerability on a well-known DApp seems to be more difficult. Furthermore, a vulnerability on a less popular DApp can still lead to huge damage.

B) Analysis of programming languages used

Programming Language	Number of DApps
TypeScript	52
JavaScript	29
Unknown	15
Python	3
Cadence	1

Table 1: Different programming languages used by top 100 DApps

The table 1 shows the different languages mainly used to develop the top 100 DApps. More than a half are written using TypeScript and a third are written with JavaScript. As TypeScript and JavaScript are really similar languages, the analysis will focus on this two languages. It represents a large majority of the most popular DApps.

15 DApps have an unknown programming language because it was not possible to find their source code. Finally, only 4 of the top 100 DApps are not written in JavaScript or TypeScript for sure. This is why our analysis does not include DApp not written in JavaScript or TypeScript.

C) Blockchain

1. What is a blockchain?

A DApp is an application based on a Blockchain which can be described as a list of blocks (chain of blocks). These blocks are linked with cryptographic methods. Each one has the hash, which represents the data of a block, of the previous block. It also contains a timestamp and the information of a transaction that had been made.

A blockchain is a digital ledger that operates in a decentralized and distributed manner, accessible to the public. It's designed to record transactions across multiple computers in a way that prevents retroactive changes without modifying all subsequent blocks and gaining agreement from the network.

A blockchain may be used, for instance, to process payment, to monitor supply chains, to share data, for copyright and royalties protection.

2. Wallet on a blockchain

One of the most common application of Blockchain is the management of cryptocurrency wallet. The most used wallet is MetaMask, another well-known one is Binance. The goal of these applications is to enable users to store cryptocurrency and transfer them. It provides an interface to interact with the blockchain.

MetaMask is a browser plugin that acts as an Ethereum wallet, and easy to use as the figure 2 depicts. By connecting to MetaMask to Ethereum-based DApps, users can spend their coins in games, stake tokens in gambling applications, and trade them on decentralized exchanges (DEXs). It also provides users with an entry point into the emerging world of decentralized finance.

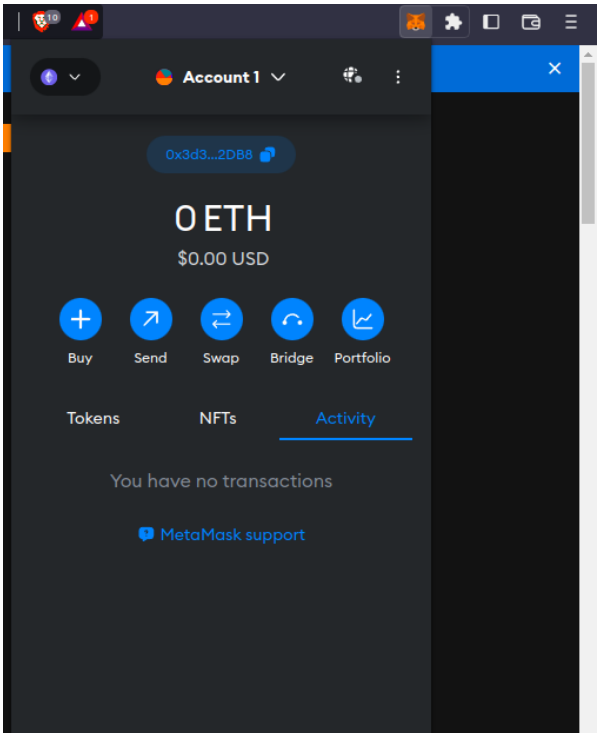

Figure 2: Metamask extension

MetaMask provides an API to integrate their wallet onto other DApps.

D) API used in DApps

1. MetaMask API

The first step was to analyze the MetaMask API in order to determine the most sensible ones. The analysis is based on the official MetaMask documentation [?] and on the MetaMask API playground [?].

As for the first documentation, it is very scarce, since it only collects some very select methods. Among these methods are the restricted methods, which are the methods that require permissions. Because of this, the methods in this documentation are very limited, and many are not well-developed.

Besides, the Metamask test page contains almost all available methods. This page also briefly explains the functionality of the methods, structure of the input and response parameters. Moreover, it also describes which of them are mandatory and which are optional. However, since it is not the official documentation webpage, it may not be up-to-date, and the results must be double checked.

The analysis has been made using a spreadsheet to gather the results as the figure 3 illustrates. It includes the different parameters used by the APIs with the distinction of mandatory and optional parameters. Then, the next column, *Permission / Confirmation* indicates whether the API requires a user confirmation via a pop-up or not. This column has some more details in another column. The column *Return* shows the type of the object returned by the API.

Finally, the column *Differences* explains what is different from the MetaMask documentation to the Ethereum Improvment Proposals (EIP) corresponding to this API. The EIPs are the standards for blockchain interaction API and will be discussed later.

	A	C	D	E	F	G	H	
1	**API name**	**Required Parameters**	**Optional Parameters**	**Permission/ Confirmation**	**Return (Doc)**	**permissions comment**	**Differences**	**Loca**
2	wallet_addEthereumChain	ChainId: string	- blockExplorerUrls: Array (string) - chainName: String - iconUrls: Array (string) - nativeCurrency: Object - rpcUrls: Array (string)	confirmation needed	null / error code	Approve or Cancel + switching network confirmation if needed	Minor: Both the EIP and metamask docs have the same specification of this method. However, in the EIP, for security reasons, they state that the metadata (the optionals parameters) should always be verified and therefore, used. The EIP also states that a wallet MAY require any other of the optional parameters.	GitH
3	wallet_switchEthereumChain	ChainId: string		confirmation needed	null / error code	Switching network confirmation or cancel. **All pending confirmations will be cancelled if aproved**	Minor: EIP states that this method can be performed without user acceptance, but they do not recommend it.	GitH
4	wallet_requestPermissions	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.	Defir https blob/ Use https tree/
5	wallet_getPermissions				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}	GitH

Figure 3: Metamask API analysis

2. Ethers.js and Web3.js API

Ethers.js provides also many APIs to interact with the Ethereum Blockchain and its ecosystem. It was originally designed for usage with ethers.io and has since expanded into a more general-purpose library. The Ethers.js documentation [**?**] is more complete than the MetaMask documentation [**?**] what simplifies the analysis.

Web3.js is the other major APIs provider to interact with the blockchain. It identifies as a collection of libraries that allow the user to interact with a local or remote ethereum node. As for Ether.js, the Web3.js documentation [**?**] is also very complete and provides information for every method that is possible to use.

In this project, we have not studied the security of these two libraries, and we chose to focus our work on the Metamask API. Since it is possible to link the MetaMask portfolio with applications developed with Ether.js and Web3.js, by studying MetaMask, we also studied the latter two libraries. Because of this, the documentation has been made method by method, as for MetaMask, with mandatory parameters, optional parameters, method return, etc.

The most important thing is to document the methods that correspond to those of Metamask. Therefore, in the spreadsheet, three sheets have been created (one for Metamask, one for Ether.js and one for Web3.js), and the column numbers match the methods. That is, the method of Metamask in one particular row, corresponds to the same methods, but from EtherJS and from Web3JS in the same row.

E) EIP

1. General concept

EIPs, which stands for Ethereum Improvement Proposals [**?**], play a crucial role in the evolution of the Ethereum blockchain. They are essentially sets of guidelines that propose and describe potential enhancements, additions, or changes to the Ethereum network's functionality. These proposals can cover a wide range of topics, such as introducing new features, improving existing processes, or suggesting updates to the overall architecture.

Each EIP is like a detailed blueprint, providing technical specifications and documentation for the proposed alteration. This documentation is essential for the Ethereum community to understand the scope, purpose, and implementation details of the suggested change. EIPs is similar to a source of truth, serving as the authoritative reference point for anyone who wishes to comprehend the intricacies of a particular proposed improvement.

One of the significant aspects of the EIP process is its decentralized nature. It encourages collaboration and input from various stakeholders in the Ethereum ecosystem. Developers, researchers, users, and other community members can engage in discussions, provide feedback, and contribute their expertise to refine and validate the proposal. This collaborative effort helps ensure that proposed changes are well-vetted and aligned with the broader Ethereum community's goals and values.

The EIP process also serves as the foundation for discussing and implementing network upgrades. When a proposal gains significant consensus and is thoroughly reviewed, it can become part of a network upgrade, often referred to as a hard fork. These upgrades can introduce substantial changes to the Ethereum protocol, enhancing its capabilities, performance, and security.

Furthermore, EIPs provide a framework for establishing application standards within the Ethereum ecosystem. By proposing standardized processes or conventions, EIPs contribute to a more cohesive and interoperable environment for DApps and Smart Contracts. This standardized approach simplifies development and enhances the overall user experience by ensuring consistency across various applications built on Ethereum.

In essence, EIPs are the backbone of Ethereum's continuous evolution. They embody the collaborative spirit of the Ethereum community, ensuring that proposed improvements are thoroughly examined, technically sound, and aligned with the platform's overarching vision of a decentralized and innovative blockchain ecosystem.

2. EIP721

The EIP721 [?] introduces a standardized API that enables the integration of Non-Fungible Tokens (NFTs) into Smart Contracts. This established standard furnishes fundamental functionality for monitoring and facilitating the transfer of NFTs.

This EIP is inspired by the ERC-20 token standard, defined in the EIP20 [?]. The introduction of the ERC-721 standard (EIP721) was necessary because the ERC-20 standard (EIP20) was not suitable for representing non-fungible assets. Fungible tokens are essentially identical and can be exchanged on a one-to-one basis, making them ideal for representing divisible assets like currencies. However, non-fungible assets are fundamentally distinct and unique, and they required a different standard to accurately represent their ownership, attributes, and individuality.

EIP721 addressed this need by providing a standardized way to create and manage non-fungible tokens, allowing developers to accurately represent ownership of unique assets in a consistent manner. This has opened the door to a wide range of applications such as digital art marketplaces, virtual collectibles, decentralized games, and more, where the uniqueness and individual characteristics of each asset are essential.

3. EIP712

The EIP712 [?] defines standards that allow developers to specify the structure of data in a way recognizable by both humans and machines. This structured data can include various fields and types of information, and it's often used to represent specific interactions, transactions, or messages that need to be signed by users.

One of the key benefits of EIP712 is that it helps prevent certain types of attacks, like Replay Attack, which will be discussed later, by ensuring that the signed data includes contextual information that uniquely identifies the transaction. This makes it harder for attackers to reuse signed data in different contexts.

The link between EIP721 and EIP712 comes into play when dealing with NFTs created according to the ERC-721 standard. Since NFTs represent unique and often valuable assets, their transactions need to be securely authorized and verified.

EIP712 is commonly used in the context of NFTs to define and structure the data

associated with token-related actions, such as transferring ownership of an NFT from one address to another. By structuring this data using the EIP712 format, it becomes easier to generate and verify digital signatures that prove the authenticity and authorization of transactions involving NFTs.

In essence, EIP712 enhances the security and trustworthiness of interactions involving NFTs (such as transfers) by providing a standardized way to structure and sign data, which can be essential in preventing unauthorized or malicious activities.

F) Signing methods in MetaMask

1. Importance of signing methods

In the blockchain, the implementation of different signature methods is necessary. All information that needs to be transmitted through this blockchain is signed for the security and integrity of the data, and to confirm that the sender is the expected one. Over the years, new and more secure forms of signatures have been discovered. Metamask has implemented different signature methods over time.

While signing methods and wallet applications like MetaMask are designed to enhance security and user control in the world of blockchain and cryptocurrencies, they can also introduce vulnerabilities if not used or implemented correctly. This is the reason why the project focuses on signing methods provided by MetaMask. Their implementation in DApps is a crucial point of security.

2. eth_sign

The first method that had been implemented by MetaMask is the least secure. Indeed, in the popup presented to the user when confirming the signature, the message is not readable, as the figure 4 shows.

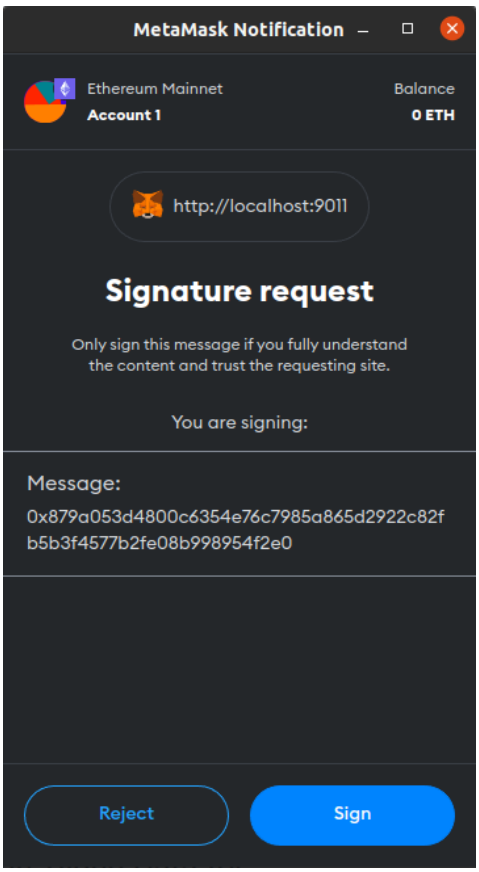

Figure 4: eth_sign pop-up message

Therefore, many attacks were based on presenting the user with this signature. As it is not readable, they sign it thinking that it is a legitimate transaction, whereas the application makes them sign something else. As can be seen in the following image, the message to be signed is a hexadecimal string that arises from hashing the signature, so the user cannot understand it.

When this study was initiated, this method was not recommended. Now, Metamask itself has listed this signature method as deprecated and prevents user from using it. As the figure 5 illustrates, the user who wants to sign a message with this API needs to activate it in MetaMask advanced settings. They also have had a warning to make sure people are aware of the risk of this signing method.

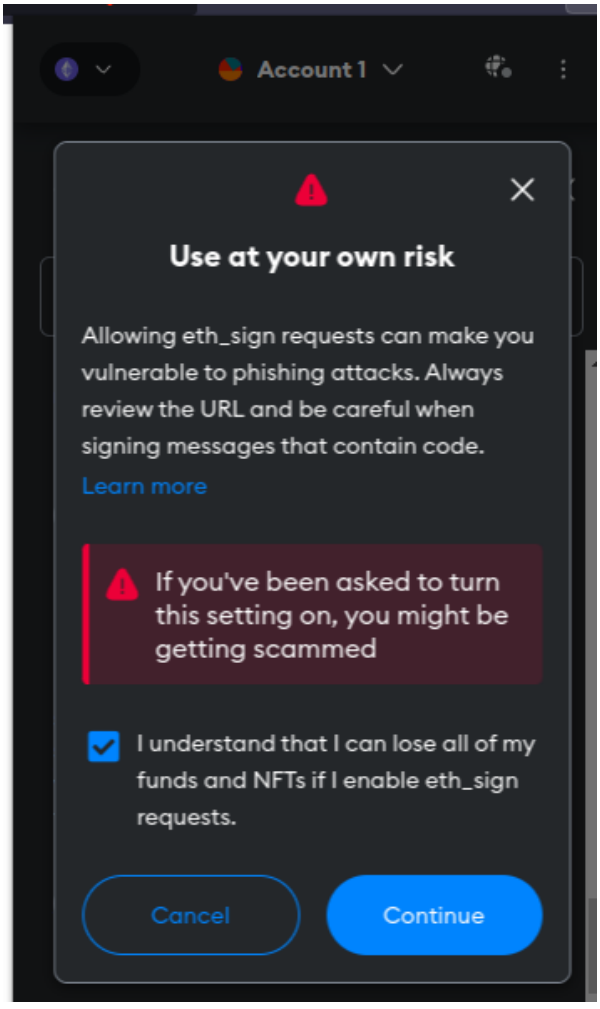

Figure 5: eth_sign warning message when trying to enable this API

In general, applications already use the other more modern methods, so the application will work without any problem. However, many DApps still use the *eth_sign* method as a signature method what breaks some functionalities of those websites.

3. personal_sign

The *personal_sign* method was introduced by Metamask to fix the signature message not being readable. This function is the easiest way to implement this. The main problem with this method, since the message is readable, is that it makes it less efficient to be processed in the blockchain as the message can be quite long. This specific method is not very efficient. Even so, many applications still use this method because it is basic. The figure 6 shows the appearance of the message presented to the user when executing this method.

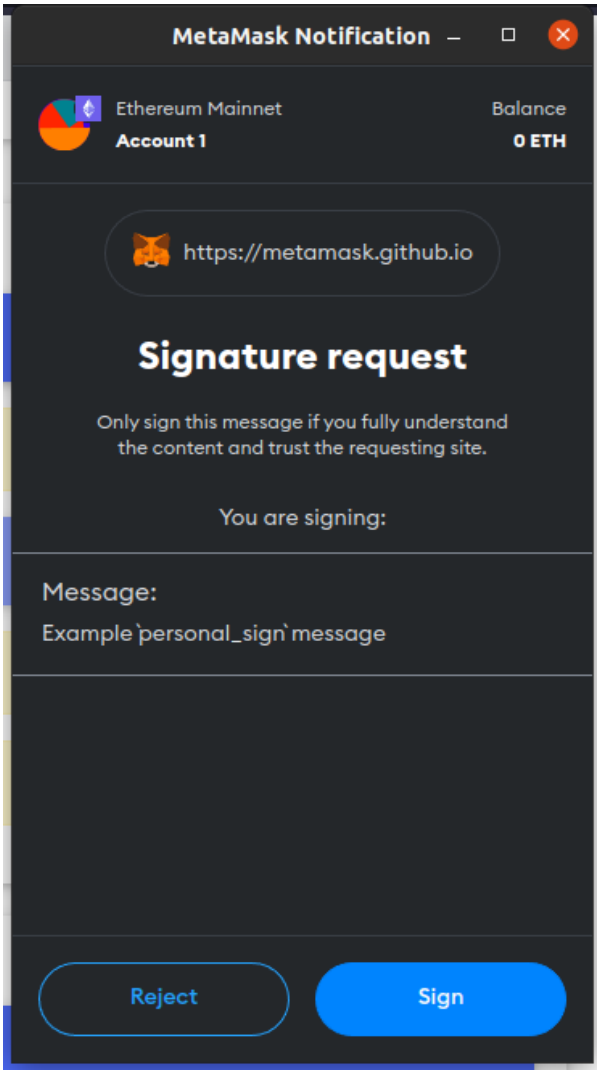

Figure 6: personal_sign pop-up message

4. eth_signTypedData

The method *eth_signTypedData* was introduced by EIP712. The EIP712 specification changed several times, resulting in multiple versions of *eth_signTypedData*.

As the figure 7 shows, the pop-up message of *eth_signTypedData* represents structured data what makes it more human readable. Moreover, being structured data makes it easy to compute for the blockchain as it can be encoded efficiently. It also cannot impersonate Ethereum transactions.

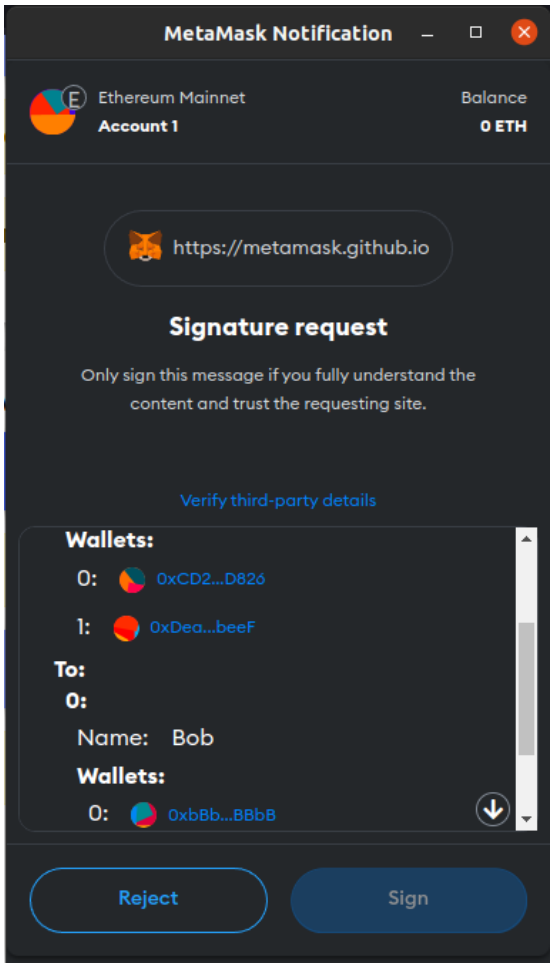

Figure 7: eth_signTypedDataV4 pop-up message

The first version of this API (*eth_signTypedData_v1* or *eth_signTypedData*) was subject to some attacks, such as Replay Attacks. Version 3 introduced some mechanisms to prevent from those attacks, like the field domain inside the message to sign (more precision will be given later). Finally, the last and only version recommended to use by MetaMask is the v4, which includes full support of arrays and recursive data structures. The missing v2 represents an intermediary design that the Cipher browser implemented.

We chose to focus on *eth_signTypedData* because *eth_sign* is already deprecated and not recommended. Furthermore, as *personal_sign* is not very efficient and *eth_signTypedData* is more complex, our analysis focuses on this API.

G) Operation of eth_signTypedData

1. Method arguments defined in EIP712

EIP712 defines several measures to enhance the security of signing methods. The one that interests us the most is the inclusion of a field named *domainSeparator*. This field includes four or five items, because the last is optional, variable:

- name: the user-readable name of signing domain, usually the name of the DApp.
- version: the current major version of the signing domain.
- chainId: the user-agent should refuse signing if it does not match the currently active chain.
- verifyingContract: the address of the contract that will verify the signature.
- salt (optional): a disambiguating salt for the protocol.

The purpose of these fields is to make the signature linked with the DApp. Before, the same message signed by the same person on different DApps had the same signature. Now, with, for instance the name of the DApp, the signature differs on two different DApps even if the message signed is the same.

The purpose of *chainId* is to differentiate different blockchain of the same DApp. Indeed, some huge DApp uses several blockchain. The value of *verifyingContract* prevents an attacker from using a malicious contract to verify the signature.

2. Creation of a test Dapp

To analyze more precisely how this method was working, the best solution is to implement a DApp. This allows trying every possibility for the argument which can be given to the method. This step is crucial because the MetaMask documentation is not precise enough and does not indicate which field can be omitted or not.

Determining which field is mandatory and which one is not plays a pivotal role in the analysis. Indeed, the majority of the fields required by *eth_signTypedData* plays a role in the security of the signing method. Therefore, if malicious DApps are able to omit some verification field, it can lead to unsafe behaviour.

Figure 8: DApp created to test eth_signTypedData

The figure 8 is a picture of the DApp created that has enabled us to understand how MetaMask handles arguments. We can try to sign a message with different structured message and then analyze the answer.

3. Mandatory fields

The figure 9 shows an example of a payload as defined by the EIP712.

There are four mandatory sections:

- domain: this is the *domainSeparator* that we discussed earlier.
- message: this field represents the useful data that we want to transfer.
- types: this defines the types of the different fields present inside the payload.
- primaryType: this gives information about the message sent.

Figure 9: Example of message sent using eth_signTypedData

The analysis has highlighted some constraint about those fields. First, every field included in the payload must be defined in *types*. Otherwise, MetaMask would ignore the missing field.

Then, *primaryType* must match a type defined in *types*. MetaMask gives an error if this condition is not met.

The *message* and *domain* fields only need to contain every field declared in *types*.

This means that only the four sections are mandatory but can be empty, except for *primaryType* and *types* which need at least one value. Nonetheless, *domain* may be an empty string and MetaMask will accept the payload. Due to this, the DApp that uses *eth_signTypedData* is able to make the user sign a message without *domainSeparator*. Nevertheless, we explained earlier that *domainSeparator* prevents from different attacks, hence, this particular point is a vulnerability in *eth_signTypedData*.

H) Analysis of signing methods in DApps

1. Finding every usage of signing methods

After finding a vulnerability in *eth_signTypedData*, the next step is to find which DApps uses this method. Moreover, we also want to find the usage of the other signing methods. Therefore, a bash script, available in the appendix A, has been developed.

The principle of the script is to download every GitHub repository from a list, and then search within every file for a particular string. In our case, we want to search for *eth_sign*, *personal_sign* and signTypedData (not *eth_signTypedData*, because sometimes it shadows some interesting match). Once we find the string, we write in an output file, the line and file corresponding to the matching string.

Then, thanks to other bash and python scripts, the results are formatted to a csv file which is easy to import in a spreadsheet to make the results more human-readable. The figure 10 shows the spreadsheet in which the results are stored.

Dapp name	Language	Source code link	Presence of "eth_signTypedData"	Presence of "eth_sign"	Presence of "personal_sign"
Uniswasp V3	TypeScript	ithub.com/Uniswap/cc	32 (usage 80)	Not found	Not found
Aave Protocol	TypeScript	github.com/aave/prot	276	Not found	Not found
Curve	TypeScript	/github.com/curvefi/cu	Not found	Not found	Not found
PancakeSwap	TypeScript	om/pancakeswap/par	213	Not found	Not found
DODO	TypeScript	ithub.com/DODOEX/	Not found	Not found	Not found
Compound	TypeScript	om/compound-financ	138	Not found	Not found
Venus Protocol	TypeScript	ub.com/VenusProtoc	138	Not found	Not found
Uniswasp V2	TypeScript	ithub.com/Uniswap/cc	32 (usage 80)	Not found	Not found
Alpaca Finance	TypeScript	m/alpaca-finance/bsc	Not found	Not found	Not found
Metamask Swap	JavaScript	om/MetaMask/meta	392	280	320
Biswap	TypeScript	'github.com/biswap-o	Not found	Not found	Not found

Figure 10: Spreadsheet which stores the results of signing methods usage in DApps

2. Collecting more DApps

Analyzing the top 100 DApps, gives a good overview of signature methods usage. However, as mentioned earlier, we cannot limit the analysis to those DApps. Therefore, it is necessary to find a way to collect more applications.

GitHub gives an API [?] that allows getting the results of a search. Using the API makes possible to develop a script to automate the finding of new DApps.

However, GitHub limits to 100 repositories that can be found by call to the API. The solution found is to get the first 100 results, and then get the next 100 until there is no more result available.

This has highlighted another limitation to GitHub API. Indeed, only the first 1000 results can be accessed no matter the filter we use. Nonetheless, this script has enabled to collect more than 2,000 other DApps, using different queries. Then, the duplicate repositories are eliminated with another bash script.

Those scripts have enabled to make the analysis a lot more solid and thorough.

The figure 11 shows the main part of the bash script that calls the GitHub API. Using the filter *per_page*, it is possible to select different range of 100 repositories.

```
echo "All results for Github repositories $1" | tee "$outputFile"

perPage=100

i=10

while curl -sL "https://api.github.com/search/repositories?q=${1// /+}+language:$language&per_page=$perPage&page=$i" | jq -r ".items[].html_url" >>"$outputFile" 2>/dev/null; do
  ((i++))
  echo "Just wrote first $((i * 100)) results"
done
```

Figure 11: Bash script to collect GitHub repositories from a query

3. Usage of signing methods

We have implemented a bash script, available in appendix B, that combines all the different other scripts mentioned above. This script enables to automate the analysis of DApps source code to find every usage of signing methods. It also formats the result to a csv file, making the results easy to read in a spreadsheet.

After running the scripts, we obtained the results presented in table 2.

Signing method	Top 100 DApps	On 2,490 DApps
eth_signTypedData	37%	5%
eth_sign	19%	14%
personal_sign	13%	10%

Table 2: Analysis of usage of signing methods in DApps

The results of the analysis highlight several points. The first one is that signing methods are more used in top 100 DApps. It can be explained because the most visited DApps are usually the ones with more functionalities, and smaller website can delegate the signing part to other applications.

The analysis also shines a light on the fact that *eth_sign* is still a lot used by DApps even the biggest ones (the 10^{th} one has been using this method).

These results also spotlight that on less visited DApps *eth_sign* is even more used than on the top 100. The figure 12 and the figure 13, which present the repartition of signing methods usage on both scopes of the analysis, allows to emphasize this point.

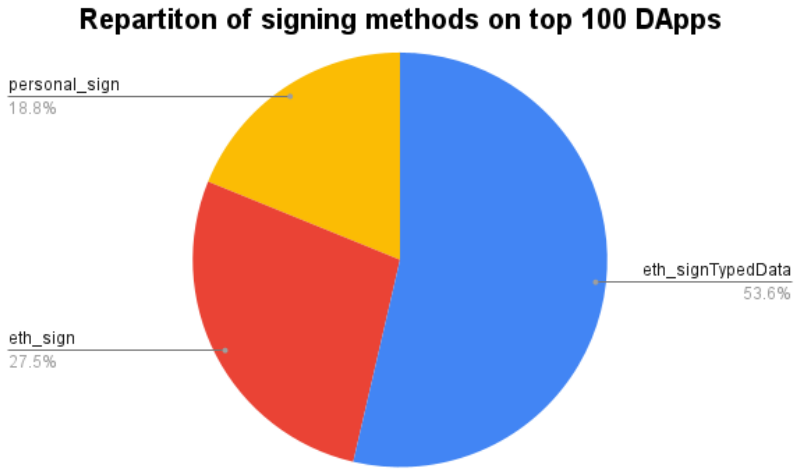

Figure 12: Repartition of signing methods on top 100 DApps

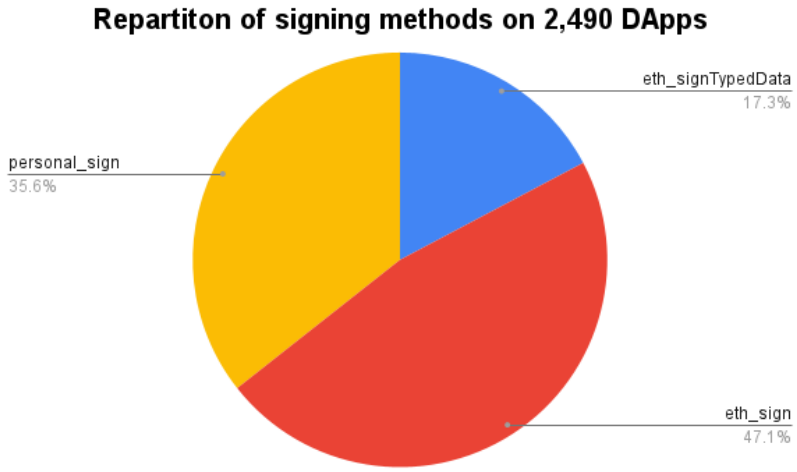

Figure 13: Repartition of signing methods on 2,490 DApps

Indeed, the two figures 12 and 13 point out the fall in the usage of the safer *eth_signTypedData* in favor to the unsecure *eth_sign*. More than half of signing methods used in top100 are *eth_signTypedData* whereas, in all DApps of the analysis, it is only around a sixth. Moreover, the usage of *eth_sign* increases from 27.5% to 47.1%. The high usage of this unsafe method reveals a vulnerability in the construction of many DApps.

I) Smart contracts implementation

1. Smart contracts role

The goal of Smart Contracts is to execute code on the blockchain. What we are interested in is the verifications of signature that are made online. As a matter of fact, every DApp needs to verify that the user who wants to make a sensible action, for instance, transferring funds. The application makes this verification on Smart Contracts deployed online. The specification of the implementation is, among others, described in EIP721.

2. ecrecover

The *ecrecover* function is a crucial element found within Ethereum smart contracts, primarily serving the purpose of signature verification for messages. When invoked, it requires specific input parameters to operate effectively: the hash of the message being verified and the signature's individual components, namely v, r, and s.

By employing intricate elliptic curve calculations, this function is capable of recovering two distinct potential public keys from the provided signature. The recovery process is facilitated by the recovery id v, which aids in pinpointing the accurate public key.

Subsequently, the Ethereum address affiliated with the signer of the message is generated through a process involving hashing the computed public key. The crux of this process revolves around confirming whether the generated address aligns with the expected address of the signer.

In practical terms, this cryptographic mechanism finds extensive application in enabling authentication and authorization protocols within smart contracts.

The proper implementation of *ecrecover* into Smart Contracts plays a pivotal role in upholding the security and integrity of the entire system. Essentially, the overarching purpose of this function lies in its ability to fortify Smart Contracts against fraudulent actions by effectively validating digital signatures.

3. Nonce and deadline

As considered with the description of EIP712 above, *domainSeparator* is a crucial point of the signature security. It helps to ensure that the signature is from the right DApp. However, it does not prevent from Replay Attacks.

Indeed, if an attacker steals a signature that gives the permission to withdraw a certain amount of money, he can validate the transaction many times. This entails a huge loss of money for the victim.

The first mechanism that DApps have introduced is *deadline*. It is a timestamp included in the signature making the signature to be expired after a certain date. This is often used for voting DApps, because it prevents people from voting after the end of the election.

The last tool used is *nonce*. A *nonce* stands for *number used once* in the realm of blockchain and cryptography. It prevents double spending by requiring each transaction to have an unique *nonce*. In Ethereum and Smart Contracts platforms, *nonces* establish transaction order and prevent Replay Attacks. In essence, *nonces* bolster security and ensure orderly processes in blockchain operations.

J) Classification of DApps

1. Purpose of classification

The next step is to analyze the Smart Contracts of the DApps. As explained earlier, they play a pivotal role by implementing the verification of the signature with or without security tools.

We focus the analysis on around 250 DApps among the most popular ones. During the work, we noticed that DApps have implemented the signature verification in very similar ways. Hence, four categories have emerged, and only a few applications have contracts that are different from others.

2. Personal Sign

The first category that has emerged is the one called *personal sign*. It is named like this because it corresponds to the implementation of *personal_sign*. As the figure 14 shows, in this implementation, there is no *domainSeparator*, no *nonce* and no *deadline*. Moreover, the prefix *Ethereum signed message* is characteristic of this category.

```
function ecrecoverAddress(bytes32 orderHash, bytes memory signature) public pure returns (address) {
    (uint8 v, bytes32 r, bytes32 s, address user, uint16 feeFactor) = decodeMmSignature(signature);

    return ecrecover(
        keccak256(
            abi.encodePacked(
                "\x19Ethereum Signed Message:\n54",
                orderHash,
                user,
                feeFactor
            )),
        v, r, s
    );
}
```

Figure 14: Ecrecover function personal sign implementation

3. Uniswap

The *Uniswap* category is the one that includes the largest number of security mechanisms. The figure 15 points out the usage of *domainSeparator* but also *nonce* and *deadline*.

This figure 15 also reveals the correct usage of *nonce*. Indeed, they increase the value after the validation of the transaction preventing this way from Replay Attacks.

```
function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external {
    require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');
    bytes32 digest = keccak256(
        abi.encodePacked(
            '\x19\x01',
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))
        )
    );
    address recoveredAddress = ecrecover(digest, v, r, s);
    require(recoveredAddress != address(0) && recoveredAddress == owner, 'UniswapV2: INVALID_SIGNATURE');
    _approve(owner, spender, value);
}
```

Figure 15: Ecrecover function Uniswap implementation

4. OpenZeppelin

OpenZeppelin [?] is a widely used framework that assists developers in building secure and reliable decentralized applications by offering pre-audited smart contracts, security best practices, and a supportive community. It has played a crucial role in advancing the security and usability of blockchain technology.

Therefore, many DApps implement the verification using OpenZeppelin. However, the figure 16 presents a function without any verification tools mentioned above. Hence, even if OpenZeppelin provides all the tools necessary for a safe utilization, one can import and use directly this function. This could lead to a risk of vulnerability because the DApp could not implement the safeguards but still think, by using OpenZeppelin, that its application is secure.

```
function recover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) internal pure returns (address) {
    // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make the signature
    // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/paper.pdf), defines
    // the valid range for s in (281): 0 < s < secp256k1n ÷ 2 + 1, and for v in (282): v ∈ {27, 28}. Most
    // signatures from current libraries generate a unique signature with an s-value in the lower half order.
    //
    // If your library generates malleable signatures, such as s-values in the upper range, calculate a new s-value
    // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v from 27 to 28 or
    // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 to v to accept
    // these malleable signatures as well.
    require(uint256(s) <= 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0, "ECDSA: invalid signature 's' value");
    require(v == 27 || v == 28, "ECDSA: invalid signature 'v' value");

    // If the signature is valid (and not malleable), return the signer address
    address signer = ecrecover(hash, v, r, s);
    require(signer != address(0), "ECDSA: invalid signature");

    return signer;
}
```

Figure 16: Ecrecover function OpenZeppelin implementation

5. Compound

The last category is named *Compound* after the most famous DApp that implements the signature verification this way.

The main difference illustrated by the figure 17 is the absence of *nonce* and *deadline*. The DApps in this category does not have every necessary tool to protect the user from malicious behavior. Even if it uses a *domainSeparator*, the missing *nonce* constitutes a vulnerability.

```
function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s) public {
    bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)), getChainId(), address(this)));
    bytes32 structHash = keccak256(abi.encode(BALLOT_TYPEHASH, proposalId, support));
    bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
    address signatory = ecrecover(digest, v, r, s);
    require(signatory != address(0), "GovernorAlpha::castVoteBySig: invalid signature");
    return _castVote(signatory, proposalId, support);
}
```

Figure 17: Ecrecover function Compound implementation

6. Different

Finally, all the DApps in this category have no similarity with the others. It does not impact the classification because, as the next section will show, this category does not contain a large number of DApps.

The figure 18 points out the difference of this implementation from the others. Indeed, the hash given to the *ecrecover* function only contains *commitLastBlock* and *commit*. This allows the attacker to reproduce the signature verification on its malicious DApp leading to a potential loss of money or item.

```
function placeBet(uint betMask, uint modulo, uint commitLastBlock, uint commit, bytes32 r, bytes32 s) external payable {
    // Check that the bet is in 'clean' state.
    Bet storage bet = bets[commit];
    require (bet.gambler == address(0), "Bet should be in a 'clean' state.");

    // Validate input data ranges.
    uint amount = msg.value;
    require (modulo > 1 && modulo <= MAX_MODULO, "Modulo should be within range.");
    require (amount >= MIN_BET && amount <= MAX_AMOUNT, "Amount should be within range.");
    require (betMask > 0 && betMask < MAX_BET_MASK, "Mask should be within range.");

    // Check that commit is valid - it has not expired and its signature is valid.
    require (block.number <= commitLastBlock, "Commit has expired.");
    bytes32 signatureHash = keccak256(abi.encodePacked(uint40(commitLastBlock), commit));
    require (secretSigner == ecrecover(signatureHash, 27, r, s), "ECDSA signature is not valid.");
```

Figure 18: Ecrecover function different implementation

7. Repartition of DApps into this classification

The table 3 summarizes the main characteristics of the different categories exposed previously.

Smart contract	***domainSeparator***	***nonce***	***deadline***
Uniswap	Yes	Yes	Yes
OpenZeppelin	No in *ecrecover* function	No in *ecrecover* function	No in *ecrecover* function
Compound	Yes	No	No

Table 3: Summary of differences between Smart Contract categories

The figure 19 represents the distribution of the different categories.

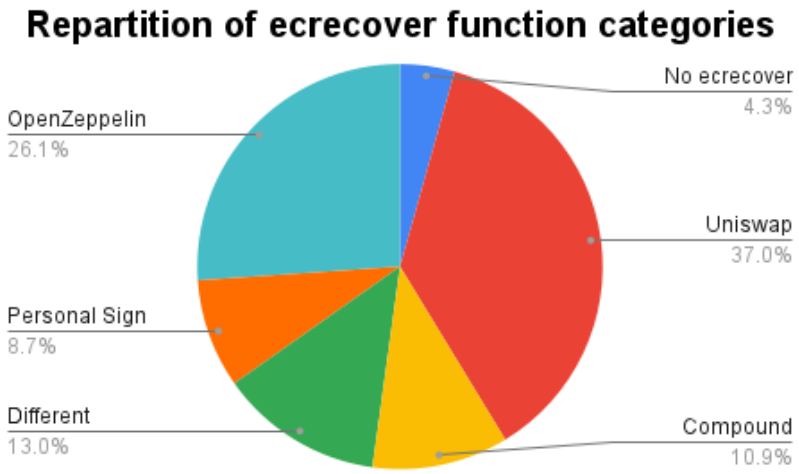

Figure 19: Repartition of ecrecover function categories

It highlights that 37% of the DApps analyzed are *Uniswap* which is the most secure category. However, 26% of the DApps can be unsafe with a wrong usage of *OpenZeppelin* library.

As mentioned before, the *different*, *personal sign* and no *ecrecover*, which signifies that the Smart Contract just does not have any usage of *ecrecover* function, represents only 26% of the DApps.

Finally, about 11% of the DApps are part of the *Compound* category which represents a vulnerability. This number is not insignificant and has to be reported.

VI. Environmental and social impact

A) Personal impact

As the project was almost only online, there was no commuting to work and no professional trip. This impact is therefore non-existent or negligible.

I do not consider the first trip to go to the USA and the return to France because even without the project, those journeys would have been done.

Most of my personal impact was the consumption of the equipment used for the project. The only piece of equipment used was my laptop.

According to the website EcoInfo and their tool EcoDiag [?] my equipment has an impact corresponding to 101 kgCO2/year as the figure 20 reckons.

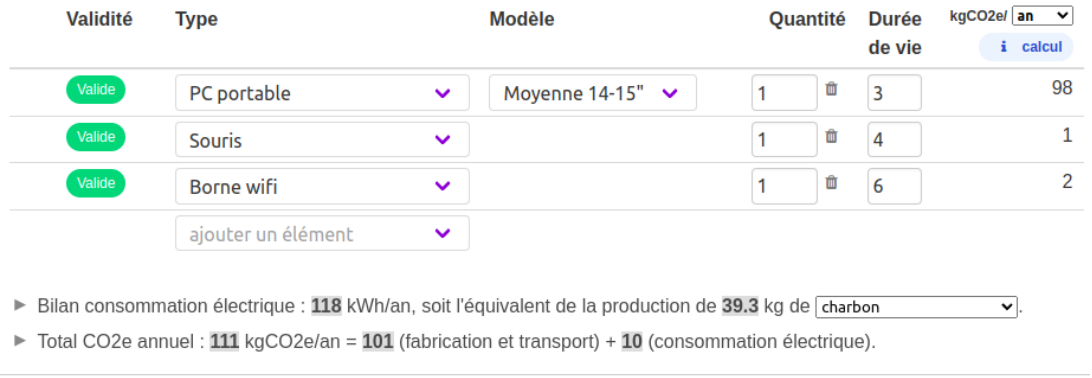

Figure 20: Environmental cost of equipment used during the project

Nevertheless, the electricity consumption impact cannot be used because the impact of electricity is different in the USA from France.

In Illinois, electricity represents 0.855 pounds of CO2 emissions per kWh which means 0.388 kgCO2/kWh. The figure 20 proposes annual consumption of 118 kW/year what means that the electricity consumption impact is 46 kgCO2/year.

Therefore, the total cost is 147 kgCO2/year. As the project lasts around 220 days, the personal impact of the project is 89 kgCO2.

It can be noticed that electric consumption plays a larger role in the impact in the USA rather than in France.

B) Global impact

1. Environmental impact

The energy consumption associated with some blockchain networks has raised concerns due to its environmental impact. Energy usage in blockchain technology is primarily attributed to the computational processes required for consensus mechanisms, transaction validation, and maintenance of the distributed ledger.

However, the project analyzes blockchain technology but does not run these technologies. Therefore, energy consumption of DApps are not linked with the project. Even if a DApp has been created for the project, it was running locally, and its impact has already been discussed in the personal analysis. The calls made to MetaMask represent a negligible environmental impact.

The location of the team is a consideration that can be made. As we saw previously, between France and the USA, electricity represents a very different number of emissions.

The team was located in Illinois, and it can be interesting to look for the difference between this state electricity impact and the US average one.

The Illinois net electricity generation by source is 7% natural gas, 30% coal-fired, 54% nuclear (most in the nation) and 10% renewables, according to Illinois Environmental Council [**?**], a governmental institution. However, in the United States, according to the U.S. Energy Information Administration [**?**], the net electricity generation by source is 39.8% natural gas, 19.6% coal-fired, 18.2% nuclear and 21.5% renewables. This makes the US average electricity source emissions of 0.371 kgCO2/kWh whereas it is a bit higher in Illinois with 0.388 kgCO2/kWh.

Enhancing security may sometimes mean increasing the consumption of applications. Indeed, if there is a need to make more verification, the scripts will run longer and therefore consume more energy.

In the context of this project, we noticed that *personal_sign* was less efficient than *eth_signTypedData*. However, the analysis has highlighted that many DApps, even in the most popular ones, uses *personal_sign* over its more efficient alternative. As those DApps can be the source of huge traffic, even a slight improvement of performance can lead to massive saves of energy.

For instance, OpenSea uses this method and has around 10,000 active users per day. Therefore, thanks to the work done on this project some users could prefer use applications that does not uses the *personal_sign* method.

However, the beneficial impact induced by the project is not quantifiable but seems to be negligible.

As the project has no tangible application in industry, its environmental impact is very low. The analysis has developed tools, but these are not designed for a massive usage. Furthermore, the equipment required by the project was only composed of a computer and its impact has been calculated on the previous section.

Finally, as the team was composed by four people, the carbon footprint of the project is around 400 kgCO2. This impact represents the carbon footprint of a French person during 13 days.

2. Societal impact

By providing users with information about the security levels of DApps, the project can enhance user confidence in using blockchain applications. Users who are aware of the potential vulnerabilities and risks can make more informed decisions, leading to a safer and more trusted environment for blockchain adoption.

The focus on vulnerabilities also educates users about the potential security risks associated with DApps. This increased awareness can extend beyond the tools developed, promoting a culture of cybersecurity consciousness among blockchain users and developers.

As developers become more aware of the vulnerabilities highlighted by the project, they may encourage adopting better security practices during development. This could lead to the overall improvement of security, reducing the prevalence of vulnerabilities.

A more secure DApp environment can mitigate the risk of exploitation, theft, and unauthorized access. This can lead to fewer incidents of financial loss suffered by users and businesses. Indeed, exploitation and theft often target end-users who might unknowingly interact with compromised DApps. By providing users with information about the security status of DApps, the project empowers them to make informed decisions. Users can avoid using DApps with known vulnerabilities, minimizing their exposure to potential threats.

About the privacy policy, all the data collected by the tools was publicly accessible and was not sensible. Moreover, the tools only collect data about applications or websites that do not contain personal data.

Finally, the analysis plays a crucial role in reducing exploitation within the blockchain ecosystem by identifying vulnerabilities, promoting early detection and remediation, and enhancing user awareness and protection. By addressing these issues, the project contributes to a more secure and trustworthy environment for blockchain applications and transactions.

C) Illinois Institute of Technology policy

1. Sustainability Integration in Curriculum

As Illinois Institute of Technology is a university, the most obvious positive action that it can lead is educating people about environment and social concerns.

The incorporation of sustainability principles, concepts, and practices across various academic disciplines and courses. The goal is to educate students about the interconnectedness of environmental, social, and economic systems, and to equip them with the knowledge and skills needed to address complex global challenges in a sustainable manner.

However, this action can suffer from potential negative aspects. Not all educators may be well-versed in sustainability concepts or just may not want to integrate these issues into their classes. This could result in inconsistent or insufficient coverage of sustainability topics in certain courses.

If not properly managed, adding sustainability content to existing courses might result in information overload for students, potentially diluting the overall learning experience. Students could question the relevance of the learning, and even doubt about the necessity to take actions.

Integrating sustainability into the curriculum is a significant step towards fostering environmentally and socially responsible citizens. While there are potential challenges, such as faculty adherence and curriculum management, the positive aspects outweigh the negatives. Proper training for educators and ongoing assessment can help ensure that sustainability integration effectively equips students with the knowledge and mindset needed to address the complex challenges of our world.

2. Energy-efficient practices

One of the main challenges to address is the energy consumption. This action involves adopting technologies, behaviors, and strategies that aim to minimize energy consumption while maintaining or improving functionality. These practices are crucial for reducing the environmental impact of energy production, conserving natural resources, and mitigating climate change.

Energy-efficient practices directly contribute to lowering greenhouse gas emissions. By consuming less energy, institutions like the Illinois Institute of Technology can help decrease their carbon footprint and combat climate change. They announce that they develop energy performance contracts on campus to reduce wasted energy through higher efficiency equipment, more building controls. They want to shift loads using geothermal heating and cooling, using daylight and passive strategies.

Nonetheless, no document with clear objectives in duration and in energy consumption reduction could have been found. Moreover, natural gas is used as a source of energy for heating buildings and generating hot water. Although natural gas is relatively clean burning fossil fuel, it is not a renewable resource leading.

However, as many places in the United States, every building is equipped with air conditioning. IIT could increase the temperature, which is set really low in my experience, to significantly reduce the energy consumption. It can use air conditioning less often and only in necessary places.

VII. Conclusion

A) Project conclusion

The analysis of the usage of signing methods has revealed that a lot of DApps are subjects to the vulnerability of *eth_sign*. Furthermore, even if the usage of this unsafe method occurs more often in less famous DApps, several famous ones are vulnerable because of the usage of this method.

Nonetheless, using a safer method such as *eth_signTypedData* does not prevent from having vulnerabilities. Indeed, the *domainSeparator* field, normally used to prevent someone from using a signature in another DApp, is not verified correctly by MetaMask. This can induce developers to not include every security field introduced by EIP712.

Moreover, some DApps implements the signature verification without using a mechanism that makes the signature unique. The *Compound* category and its non-negligible number of DApps has shown an unsecure behavior in several applications.

Finally, the OpenZeppelin category also draws attention because of its potential lack of security when developers does not use the library the right way.x

B) Future work

Collecting even more DApps is a way to extend the analysis. One way would be to overpass the limitation of the GitHub API. This may be possible by selecting the results from a date to another date. Thereby, there would be less than a thousand results after a call to the API. Indeed, changing the date corresponds to querying another time the API. The challenge is to find the best range of date to not multiply the calls and not to have more than a thousand results.

Furthermore, several DApps seems to be particularly vulnerable. For instance, Demex and Accross protocol have drawn attention. The first one has a fixed string as *verifyingContract* inside its *domainSeparator* instead of a dynamic value corresponding to the real contract. The second one does not include any *verifyingContract* on the *domainSeparator*. These two examples shine a light on the need of verifying the value of every field of *domainSeparator*.

C) Personal conclusion

This project has been an important development for my academic and research growth. It has given me the tools to delve further into blockchain technology and cybersecurity.

Moreover, the discovery of blockchain and DApps operation has provided me with a profound appreciation for the decentralized spirit that supports these innovations. Witnessing the power of cryptographic principles and the benefits of decentralization has strengthened my perspective on the possibilities that blockchain technology can offer to various industries.

As I reflect on this project, I realize it has ignited a curiosity to continue exploring and learning. The challenges I have faced and the knowledge I have acquired have deepened my understanding and passion for this field. I look forward to pursuing further studies in a PhD program, drawing from both the technical skills and the research experience I've gained here.

Appendix A Bash script to find signing methods

```
#!/bin/bash

############################################################
# Help                                                     #
############################################################

Help() {
  # Display Help
  echo "Search for a string inside every git repository from list."
  echo
  echo "Syntax: finderGithub [-h|j|o|s] repoFile"
  echo "options:"
  echo "h     Print this Help."
  echo "j     Search for signTypedData, eth_sign and personal_sign in TS
    and JS files."
  echo "o     Output file."
  echo "s     Search for sign and ecrecover in Solidity files "
}

############################################################
# Process the input options.                               #
############################################################

while getopts "h:jo:s" option; do
  case $option in
  h) # display Help
    Help
    exit
    ;;
  j) # search in JS and TS
    regNum=1
    ;;
  o) # output file
    outputFile=$OPTARG
    ;;
  s) # search in solidity
    regNum=2
    ;;
  \?) # Invalid option
    echo "Error: Invalid option"
    exit
    ;;
  esac
done

shift "$((OPTIND - 1))"

if [ -z "$1" ]; then
  echo 'Missing repoFile argument' >&2
  Help
  exit 1
fi

if [ -z "$outputFile" ]; then
```

```
54    echo 'Missing -o option' >&2
55    Help
56    exit 1
57  fi
58
59  ############################################################
60  # Main program                                             #
61  ############################################################
62
63  workdir=$(pwd)
64
65  if [ -z "$regNum" ]; then
66    echo "
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      "
67    echo "| Search for:
            |"
68    echo "|    1: SignTypedData, eth_sign and personal_sign in TS and JS
       files   |"
69    echo "|    2: Sign and ecrecover in Solidity files
            |"
70    echo "
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      "
71
72    read -r regNum
73  fi
74
75  if [ "$regNum" -eq 2 ]; then
76    stringToSearch="sig(n|nature)"
77    stringToSearch2="ecrecover"
78    fileRegex="*.sol"
79    fileRegexExclude=":!node_modules/**"
80    fileRegexExclude2=":!build/**"
81    echo "Results for $stringToSearch and $stringToSearch2" >"$outputFile"
82  else
83    stringToSearch="signTypedData"
84    stringToSearch2="eth_sign"
85    stringToSearch3="personal_sign"
86    stringToSearch4="signTypedData_V1"
87    stringToSearch5="signTypedData_V3"
88    stringToSearch6="signTypedData_V4"
89    fileRegex="*.[tj]s"
90    fileRegexExclude=":!node_modules/**"
91    fileRegexExclude2=":(attr:!vendored)*.js"
92    echo "Results for $stringToSearch, $stringToSearch2 and
       $stringToSearch3" >"$outputFile"
93  fi
94
95  echo ""
96
97  nbLines=$(wc -l <"$1")
98  counter=1
99
100 while read -r repolink; do
101   repoTemp=$(echo "$repolink" | cut -d '/' -f 4-5)
102   repo=$(echo "$repoTemp" | cut -d '.' -f 1)
```

```
103
104   echo "======================================="
105   echo "Searching in $repo ~ ($counter/$nbLines)"
106
107   echo "======================================= $repo
       ======================================="  >>"$outputFile"
108
109   git clone -q "https://:@github.com/$repo" "$workdir/tmpGitRepo"
110   cd "$workdir/tmpGitRepo" || continue
111   {
112     echo "~~~~~~~~~~~~~~~~~~ $stringToSearch ~~~~~~~~~~~~~~~~~~"
113     git grep -I -n -i -o -E "$stringToSearch" -- "$fileRegex" "
      $fileRegexExclude" "$fileRegexExclude2"
114     echo "~~~~~~~~~~~~~~~~~~ $stringToSearch2 ~~~~~~~~~~~~~~~~~~"
115     git grep -I -n -i -o -E -w "$stringToSearch2" -- "$fileRegex" "
      $fileRegexExclude" "$fileRegexExclude2"
116   } >>"../$outputFile"
117
118   if [ "$regNum" -eq 1 ]; then
119     {
120       echo "~~~~~~~~~~~~~~~~~~ $stringToSearch3 ~~~~~~~~~~~~~~~~~~"
121       git grep -I -n -i -o -E -w "$stringToSearch3" -- "$fileRegex" "
      $fileRegexExclude" "$fileRegexExclude2"
122       echo "~~~~~~~~~~~~~~~~~~ $stringToSearch4 ~~~~~~~~~~~~~~~~~~"
123       git grep -I -n -i -o -E -w "$stringToSearch4" -- "$fileRegex" "
      $fileRegexExclude" "$fileRegexExclude2"
124       echo "~~~~~~~~~~~~~~~~~~ $stringToSearch5 ~~~~~~~~~~~~~~~~~~"
125       git grep -I -n -i -o -E -w "$stringToSearch5" -- "$fileRegex" "
      $fileRegexExclude" "$fileRegexExclude2"
126       echo "~~~~~~~~~~~~~~~~~~ $stringToSearch6 ~~~~~~~~~~~~~~~~~~"
127       git grep -I -n -i -o -E -w "$stringToSearch6" -- "$fileRegex" "
      $fileRegexExclude" "$fileRegexExclude2"
128     } >>"../$outputFile"
129   fi
130
131   cd "$workdir" || exit 1
132   rm -rf "$workdir/tmpGitRepo"
133   ((counter++))
134
135 done <"$1"
```

Appendix B Bash script that automate the analysis of signing methods

```bash
#!/bin/bash

#############################################################
# Help                                                      #
#############################################################

Help() {
  # Display Help
  echo "Search for a string inside every git repositories from a search in github."
  echo
  echo "Syntax: findInGithubSearch [-f|h|l|o] query"
  echo "options:"
  echo "f     File with previous github link you don't want to search in."
  echo "h     Print this Help."
  echo "l     Specify a language."
  echo "o     Output file (mandatory)."
}

#############################################################
# Process the input options.                                #
#############################################################

while getopts "f:h:l:o:" option; do
  case $option in
  f) # file with link to search in
    fileNotToSearch=$OPTARG
    ;;
  h) # display Help
    Help
    exit
    ;;
  l) # file with link to search in
    language=$OPTARG
    ;;
  o) # output file
    outputFile=$OPTARG
    ;;
  \?) # Invalid option
    echo "Error: Invalid option"
    exit
    ;;
  esac
done

shift "$((OPTIND - 1))"

if [ -z "$1" ]; then
  echo 'Missing query argument' >&2
  Help
  exit 1
```

```
fi

if [ -z "$outputFile" ]; then
  echo 'Missing -o option' >&2
  Help
  exit 1
fi

############################################################
# Main program                                             #
############################################################

removeFirstLine() {
  tail -n +2 "$1" >"$1.tmp" && mv "$1.tmp" "$1"
}

workdir=$(pwd)

"$workdir"/scripts/bash/searchRepo.sh -o tmpSearch.txt -l "$language" "
    $1"

removeFirstLine tmpSearch.txt

if [ -z "$fileNotToSearch" ]; then
  cat tmpSearch.txt >tmpMerge.txt
else
  "$workdir"/scripts/bash/mergeFile.sh -o tmpMerge.txt tmpSearch.txt "
    $fileNotToSearch"
fi

"$workdir"/scripts/bash/finderGithub.sh -j -o tmpFinder.txt tmpMerge.txt

removeFirstLine tmpFinder.txt

"$workdir"/scripts/bash/getName.sh -o tmpNames.txt tmpMerge.txt

python3 "$workdir"/scripts/python/match_to_csv.py tmpFinder.txt tmpMatch
    .csv
python3 "$workdir"/scripts/python/add_name_link_to_search_result.py
    tmpNames.txt tmpMerge.txt tmpMatch.csv "$outputFile"

rm "$workdir"/tmpMerge.txt
rm "$workdir"/tmpSearch.txt
rm "$workdir"/tmpNames.txt
rm "$workdir"/tmpMatch.csv
rm "$workdir"/tmpFinder.txt
```