XDEFI decentralized wallet extension Architecture and Code review

Xdefi Technologies

01 April 2021
Version: 1.0

Presented by:
Kudelski Security Research Team
Kudelski Security – Nagravision SA

Corporate Headquarters
Kudelski Security – Nagravision SA
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.0
File Name:	XDEFI_review_v1.0
Publication Date:	01 April 2021
Confidentiality Level:	Confidential
Document Owner:	Mikael Björn
Document Recipient:	Emile Dubie, David Phan
Document Status:	Approved

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

EXECUTIVE SUMMARY

Kudelski Security ("Kudelski"), the cybersecurity division of the Kudelski Group, was engaged by Xdefi Technologies ("XDEFI") client to conduct an external security assessment in the form of a Architecture and Code review of the XDEFI decentralized wallet extension application.

The assessment was conducted remotely by the Kudelski Security Team from our secure lab environment. The tests took place from February 22, 2021 to March 14, 2021 and focused on the following objectives:

1. To help the Client to better understand its security posture on the architecture and implemented code.
2. To provide a professional opinion on the maturity, adequacy, and efficiency of the architecture and implemented code.
3. To identify potential issues and include improvement recommendations based on the result of our review.

This report summarizes the tests performed and findings in terms of strengths and weaknesses. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to exploit each vulnerability, and recommendations for remediation.

1.1 Engagement Limitations

The architecture and code review is based on the documentation and code provided by XDEFI. The code resides in a private repository at https://github.com/XDeFi-tech/xdefi-extension

The reviews are based on the commit hash 955a2e275d6634491e70d2c08d3bf46bd567da22

All third-party libraries were deemed out-of-scope for this review and are expected to work as designed.

1.2 Engagement Analysis

This engagement was comprised of an architecture review, implementation review, and a code review. The architecture review was based on the documentation and the information retrieved through communication between the XDEFI team and the Kudelski Security team. The architecture and implementation review concluded that the application has a sound architecture, design and the implementation is a good as expected for a browser extension application.

The code review was conducted by the Kudelski Security team on the code provided by XDEFI, in the form of a Github repository. The code review focused on the handling of secure and private information handling in the code.

As a result of our work, we identified **4 High**, **3 Medium**, **1 Low**, and **18 Informational** findings.

Almost all findings are a result of the code still being in active development and that logging and debugging creates leaks where private information is made visible to an attacker. These have already been mediated.

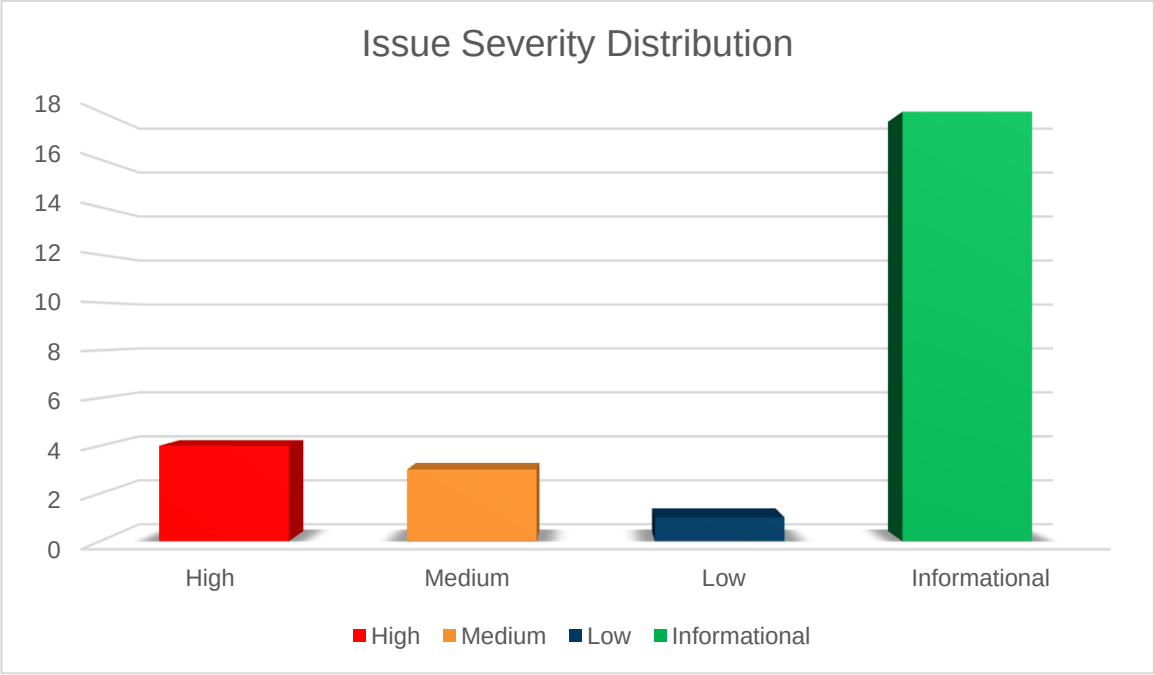

Figure 1 Issue Severity Distribution

1.3 Observations

The browser plugin has a very broad privileges to be able to inject information into other pages. This may be an attack vector for a malicious party injecting code into the plugin and then it would spread it on to the pages it addresses. This would eb handled most efficiently by checking all input picked up from other pages against the standard injection methods. Plugin reads from page -> Page inject js code into the plugin-data -> the plugin propagates this data to the pages it injects into.

The comments in the code are very few and sparse. It would be good to use some kind of standard template to define the input, output and any side effects of the code as well as the overall purpose of the code, by using more than just the name of the function or a constant.

The coding practice is very much based on a need basis and there are functions that could be modularized to be reused i.e. max-functions for transaction limits. This duplication may introduce errors as getting all places fixed may be hard.

The use of constant vs. function declaration in the files also make the code structure forced to be declaration first, use later, which may impede the logical code structure. By using functions instead, you would be able to structure the code more freely and see if there is any reuse possible being more conservative when adding code. Suggesting a refactoring of the codebase.

It is a good practice to use a good naming standard for functions. We see the same functionality use different names in different files i.e. "redirectHandler" versus "handleRedirect". Pick a verb form that makes it easy to be consistent over the codebase.

It would be a good idea to implement a consistent error handling on all pages. There is sporadic handling of errors in the Send/Tx pages but not on many others.

The Getter/Setter is implemented as default. If a constant is not to be changed it is a good practice NOT to create a setter method as that method may be abused.

Some parts of the code are in very early development stages and contains massive amounts of dead our commented code. This must be removed!

The master password is only 8 characters and there are no controls on how the password should be constructed by the user. The user should be guided to create a strong password based on best practices. As the value of the data stored behind this password is high, we strongly advice on changing this practice.

Inconsistent use of ESLINT suppressing arguments in the files. During Development none should be present to really know what you are dealing with. In production only the ones needed to not create a problem should be allowed.

Consider the usage of a server-side logging solution that would encrypt and store the logged information instead of using client side logging. This information will probably be hard to get hold of anyway for debugging purposes.

1.4 Issue Summary List

ID	SEVERITY	FINDING
KS-XDEFI-F-04	High	Private information leakage
KS-XDEFI-F-05	High	Private information leakage
KS-XDEFI-F-06	High	Private information leakage
KS-XDEFI-F-07	Medium	Information leakage
KS-XDEFI-F-11	Medium	Information leakage
KS-XDEFI-F-13	Medium	Information leakage
KS-XDEFI-F-16	High	Private information leakage
KS-XDEFI-F-23	**Low**	Information leakage

2. METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2 Methodology Flow

2.1 Kickoff

The project is kicked all of the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

2.2 Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

2.3 Review

The review phase is where a majority of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the application
2. Review of the code written for the project

3. Assessment of the cryptographic primitives used
4. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and tools, utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Cryptography

We analyzed the cryptographic primitives and components as well as their implementation. We checked in particular:

- Matching of the proper cryptographic primitives to the desired cryptographic functionality needed
- Security level of cryptographic primitives and their respective parameters (key lengths, etc.)
- Safety of the randomness generation in general as well as in the case of failure
- Safety of key management
- Assessment of proper security definitions and compliance to use cases
- Checking for known vulnerabilities in the primitives used

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

2.4 Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project as a whole. We may conclude that the overall risk is low, but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

2.5 Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes withing a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

2.6 Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These is a solid baseline for severity determination. Information about the severity ratings can be found in **Appendix C** of this document.

3. ARCHITECTURE REVIEW - XDEFI DECENTRALIZED WALLET APPLICATION

3.1 Introduction

Kudelski Security has had the opportunity to review the XDEFI application from an architecture standpoint. Based on the project's documentation as a part of the complete security review, we have reached the following conclusion based on the different aspects from an architectural standpoint.

3.2 Executive summary

Based on the XDEFI decentralized wallet application's architecture review, we can conclude that the architecture is well designed and well implemented. It also scores a high architecture fitness score.

As shown in areas of Maintainability, Testability, Reusability, and Debug-ability/Monitoring, there are areas of improvement.

Kudelski Security advises that Maintainability, Testability, and Debug-ability/Monitoring would be strengthened by improving the listed deficiencies.

3.3 Background

Software architecture is defined as

- A set of artifacts (that is: principles, guidelines, policies, models, standards, and processes) and the relationships between these artifacts, that guide the selection, creation, and implementation of solutions aligned with business goals
- Software architecture is the structure of structures of an information system consisting of entities and their externally visible properties, and the relationships among them
- A software architecture is a description of the sub-systems and components of a software system and the relationships between them
    - Sub-systems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system
    - The software system is an artifact. It is the result of the software design activity

From this definition we are able to give a overview of how the XDEFI architecture is designed and implemented.

As the application is already architectured, designed and implemented we are going to see how the architecture has been implemented, as well as how it was designed from the start.

The review will be looking at the architecture from a number of different lenses to create an as complete picture of the implementation and design, as possible.

3.4 Analysis of the architecture and implementation

This architecture review is based on the documentation and the source code in the repository, https://github.com/XDeFi-tech/xdefi-extension with the commit hash 955a2e275d6634491e70d2c08d3bf46bd567da22, provided by the XDEFI team.

3.4.1 Performance

When reviewing the architecture design specifications and the code from a performance perspective, the following is found

- The connections to the underlaying are kept open and used through the internal hidden page. Therefore, the speed of interaction between the browser extension and the application tabs will be as fast as the underlying system's connection.
- The application keeps the internal data of the extension on the internal page protected by the browser sandbox implementation. As this is something very common and crucial to most extensions of this sort, the browsers' sub-system is very well tuned to handle quick and secure communication between the browser and the application.

The conclusion is that the application has a high-performance profile for the intended use.

3.4.2 Reliability

When reviewing the architecture design specifications and the code from a reliability perspective, the following is found

- The application architecture is designed to handle a good range of errors and states.
- The implementation of the application handles the errors that may arise. The implementation makes the application well suited for the intended use.

The conclusion is that the application has a high reliability profile for the intended use.

3.4.3 Availability

When reviewing the architecture design specifications and the code from an availability perspective, the following is found

- The architecture is based on React Redux for handling the application state. This way, there is a good way to ensure the state is dealt with throughout the application.
- The decentralized design makes it also suitable as a web extension as the data may be recreated based on secure backups and transaction tracing.
- The application uses the error handling features of both the browser and the libraries included from third parties.

The conclusion is that the application has a high availability profile for the intended use.

3.4.4 Security

The application security is discussed in detail in the code review part of the report. From an architecture review standpoint, the architecture itself is as good as other browser extension wallet applications, i.e., Metamask.

3.4.5 Modifiability

When reviewing the architecture design specifications and the code from a modifiability perspective, the following is found

- By using a very modular design, the addition and removal of functionality are very easy.
- The modular design also minimizes the dependencies on other parts of the code.
- Something that may become an issue is how to handle the dependencies of 3rd party libraries. This dependency is nothing unique to this application but is a widespread problem in the industry. To handle this dependency issue, the authors of the application should implement a strategy and procedure to reduce the impact on a change in third-party libraries or other dependencies.

The conclusion is that the application has a high modifiability profile for the intended use.

3.4.6 Portability

When reviewing the architecture design specifications and the code from a portability perspective, the following is found

- As a browser extension, there are limitations of how portable it may be. This limitation is based on the way that the application is developed against the WebExtension API.
- The WebExtension API is supported by the Chromium, Firefox, and Opera families of browsers.
- The extension application is developed using Javascript/Typescript, and based on the WebExtension API, the dependence on an operating system is removed as long as one of the supported browsers is available.

The conclusion is that the application has a high portability profile if one of the supported browsers is present on the target system.

3.4.7 Functionality

When reviewing the architecture design specifications and the code from a functionality perspective, the following is found

- The intended functionality in the architecture is implemented in the code.
- As seen in the code review, some areas are still in development that still need implementation, i.e., Bitcoin support.
- Any omissions in the current implementation of functionality, based on the architecture, are because the application is still in development. The omitted functionality can be implemented later without the need to rewrite large parts of the code.

The conclusion is that the application has a high functionality profile for the intended use, and the current omitted functionality can be implemented at a later stage.

3.4.8 Extensibility

When reviewing the architecture design specifications and the code from an extensibility perspective, the following is found

- The architecture, the design, and the implementation is well suited to be extended with new functionality.
- The code's modularity allows for new tokens, chains, and crypto assets to be handled by extended functionality in the application.
- React Redux has been used to handle the application state and makes it very easy to handle any extensions to the code by providing for their needs.

The conclusion is that the application has a high extensibility profile for the intended use.

3.4.9 Conceptual Integrity

When reviewing the architecture design specifications and the code from a conceptual integrity perspective, the following is found

- The current implementation well meets the concept and idea behind the application.
- The application handles the application's vision well by providing extensions, additions, and changes without compromising the current implementation.
- Based on the discussions with the application team, we can see a good connection between the vision and the implementation.

The conclusion is that the application has a high conceptual integrity profile for the intended use.

3.4.10 Interoperability

When reviewing the architecture design specifications and the code from an interoperability perspective, the following is found

- The application uses simple data-types that are well known in a good number of languages and systems.
- Using JSON as the data interchange format, the need to handle complex dictionaries and data transformation elements is kept minimal.
- This way, the data kept and stored may be used on all platforms supported by the application's run-time environment.
- By using third-party libraries implementing the different standard handling methods for the supported blockchains, the complexity of connecting to blockchain providers is minimized.

The conclusion is that the application has a high interoperability profile for the intended use.

3.4.11 Usability

When reviewing the architecture design specifications and the code from a usability perspective, the following is found

- The way that the functionality is expressed to the user is very well implemented.
- The reason for this is that the application is in a closed beta stage in the development cycle.

- This way, user interaction errors and design flaws are dealt with expediently and effectively.
- The application's usability is as good or better as other comparable browser extension wallet applications from an architecture review standpoint.

The conclusion is that the application has a high usability profile for the intended use.

3.4.12 Maintainability

When reviewing the architecture design specifications and the code from a maintainability perspective, the following is found

- The code is well structured based on the architecture
- The code is well modularized to make it easy to extend the functionality.
- The code makes maintainability easy as the structure is clear.
- Minor issues in maintainability have been raised in the code review
- Processes and procedures to handle dependencies of third-party libraries and extensions need to be in place and documented.

The conclusion is that the application has a medium maintainability profile for the intended use.

3.4.13 Efficiency

When reviewing the architecture design specifications and the code from an efficiency perspective, the following is found

- The architecture is very clear of what part of the application is responsible for what.
- The implementation of the architecture and the design clarifies what resources shall be owned and allocated by which part.
- The relationship and communication between the different parts of the application make it easy to optimize every module's use while keeping the application integrity intact.

The conclusion is that the application has a high efficiency profile for the intended use.

3.4.14 Testability

When reviewing the architecture design specifications and the code from a testability perspective, the following is found

- The architecture makes testing on all different levels possible.
- From a functional standpoint, the following test types are possible based on the architecture
    - Unit Testing – Functionality testing on code units
    - Integration Testing – Testing the integration of different code units
    - System Testing – Testing the system as a whole

    - Interface Testing – Testing of the user interface for consistency
    - Regression Testing – Testing that reveals errors after a change
    - Beta/Acceptance Testing – testing to ensure that the quality is met
- Using automated tests for the first 5 and basing the last one on the results makes up for a very lean development model.
- From a non-functional standpoint, the following test types are possible based on the architecture
    - Performance Testing – Testing to see if the performance is adequate
    - Load Testing – Test what will happen if the application is put under load
    - Volume Testing – Test what will happen if the data load increases
    - Security Testing – Test to see if there is any security issues
    - Compatibility Testing – Test if compatibility is maintained over releases
    - Install Testing – Test how the install procedure functions
    - Recovery Testing – Test how the application can be recovered
    - Reliability Testing – Test if the application will be affected by its environment
    - Compliance Testing – Test if the application complies with regulatory standards
    - Localization Testing – Test that localization works and does not affect the functionality
- Using automated tests for the non-functional test are also possible, but at the moment, most of the non-functional tests are performed as part of the closed beta-test.

The conclusion is that the application has a medium testability profile for the intended use. This profile could be mediated by implementing automated testing to a greater extent for the functional tests. For the non-functional test, a complete testing strategy needs to be designed and implemented. Test frameworks for Static Code Analysis should be implemented as a stage in a CI/CD pipeline.

3.4.15 Reusability

When reviewing the architecture design specifications and the code from a reusability perspective, the application has no intention to be reused. The application intends to provide a specific set of functionalities outlined in the vision for the application. This vision does not include reusing the code or architectural elements for other purposes, even if this would be possible.

The conclusion is that the application has a low reusability profile for the intended use, but this is by design.

3.4.16 Ease of Deployment

When reviewing the architecture design specifications and the code from an "ease of deployment" perspective, the following is found

- When in a public release, the extension is provided through the different browser vendors' extension stores. This provision model makes it easy and reliable to deploy the extension.
- The deployment is based on a pull methodology where the application requests the installation through the vendor-provided browser-store.

The conclusion is that the application has a high "ease of deployment" profile for the intended use.

3.4.17 Ease of Administration

When reviewing the architecture design specifications and the code from an "ease of administration" perspective, the following is found

- As the application is based on decentralized architecture and implemented on top of the WebExtension API, the application's administration is based on the application design requiring zero administration from a centralized vantage point.

The conclusion is that the application has a high "ease of administration" profile for the intended use.

3.4.18 Scalability

When reviewing the architecture design specifications and the code from a scalability perspective, the following is found

- The application is based on the WebExtension API, which is implemented in a browser. A browser is, by design, very scalable.
- All dependencies on which the application builds are decentralized and, by design, extremely scalable.
- Blockchain applications are by design distributed and well suited for scalability.
- No parts in the application create any scalability problems for the system as a whole.

The conclusion is that the application has a high scalability profile for the intended use.

3.4.19 Debug-ability/Monitoring

When reviewing the architecture design specifications and the code from a debug-ability/monitoring perspective, the following is found

- The logging functionality in the code at the review was not as consistent as required by the architecture.
- The logging functionality leaked private information to surrounding environments.
- Having logging functionality client-side only is problematic as error correlation analysis is impossible to perform.
- Implementing a centralized logger functionality for application health monitoring and application intelligence is crucial for an extremely distributed system.

The conclusion is that the application has a low debug-ability/monitoring profile for the intended use.

3.4.20 Development Productivity

When reviewing the architecture design specifications and the code from a development productivity perspective, the following is found

- The development is based on Open Source projects and code.
- Development is based on well-documented APIs and implementations.
- Third-party libraries are well maintained and actively developed
- Tools for development are readily available at low cost.
- The tools available have high productivity support by implementing advanced functionality supporting the developer in all aspects of the development process.
- The number of developers used to the components and technologies used in the architecture is high.

The conclusion is that the application has a high development productivity profile for the intended use.

3.5 Summary of the architecture review

Based on the analysis of the architecture and implementation, we can conclude the following

Table 1: Architeture analysis score table

Analysis	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
High	x	x	x	x	x	x	x	x	x	x	x		x			x	x	x		x
Medium												x		x						
Low															x				x	

Based on a scoring system where High equals 3, Medium equals 2, and Low equals 1. This is then summed up and divided by the number of review lenses and normalized. This gives the overall application architecture fitness score.

For the XDEFI decentralized wallet application, the architecture fitness score is 8.5

Compared to a perfect system that would gain a score of 10.

The conclusion is that the architecture is well designed and well implemented.

4. TECHNICAL DETAILS

This section contains the technical details of our findings as well as recommendations for improvement.

Based on the initial discussions and ongoing communication with the XDEFI team, the following files containing references to masterKeys are relevant for the code review

- App.tsx
- components/TxPending/index.tsx
- config/Background/controllers/MasterController.ts
- config/Background/controllers/MasterKeystoreController.ts
- config/Background/controllers/WalletsController.ts
- containers/Authenticated/Send/BinanceSend/index.tsx
- containers/Authenticated/Send/BinanceSmartChainSend/index.tsx
- containers/Authenticated/Send/BitcoinSend/index.tsx
- containers/Authenticated/Send/EthereumSend/index.tsx
- containers/Authenticated/Settings/Account/Actions/index.tsx
- containers/Authenticated/Settings/Account/Backup/index.tsx
- containers/Authenticated/Settings/Account/index.tsx
- containers/Authenticated/Settings/index.tsx
- containers/Unauthenticated/Home/index.tsx
- containers/Unauthenticated/ImportWallet/PhraseImport.tsx
- containers/Unauthenticated/ImportWallet/StandardImport.tsx
- containers/Unauthenticated/Locked/index.tsx
- containers/Unauthenticated/MasterKey/index.tsx
- containers/Unauthenticated/MasterKey/VerifyMasterPhrase.tsx
- containers/Unauthenticated/RecoverMasterKey/index.tsx
- containers/Unauthenticated/RecoverMasterPhrase/index.tsx
- ext/dapp/connect/components/Locked/index.tsx
- ext/dapp/connect/DappConnect.tsx
- ext/dapp/multiTxMsgSign/components/BottomActions/index.tsx
- ext/dapp/multiTxMsgSign/components/SignRequested/index.tsx
- ext/dapp/transaction/DappTransaction.tsx
- routers/UnauthRouter.tsx

4.1 Private information leakage

Finding ID: KS-XDEFI-F-04

Severity: **High**

Status: **Remediated**

Description

Logger leaks private information.

Proof of Issue

Filename: config/Background/controllers/MasterKeystoreController.ts

Beginning Line Number: 169

```
logger.info(
  '#MARK - generate from keystore id: masterPhrase - ',
  masterPhrase
)
logger.info(
  '#MARK - password from keystore id: masterPhrase - ',
  pubKey.toString()
)
```

Severity and Impact Summary

Logging master key phrase leaks private information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.2 Private information leakage

Finding ID: KS-XDEFI-F-05

Severity: **High**

Status: **Remediated**

Description

Logger leaks private information.

Proof of Issue

Filename: config/Background/controllers/WalletsController.ts

Beginning Line Number: 236

```
logger.info(
  '#MARK - Import Wallet Keystore - new wallet password',
  walletPassword
)
```

Severity and Impact Summary

Logging the new wallet password leaks private information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.3 Private information leakage

Finding ID: KS-XDEFI-F-06

Severity: **High**

Status: **Remediated**

Description

Logger leaks private information.

Proof of Issue

Filename: config/Background/controllers/WalletsController.ts

Beginning Line Number: 240

```
const phrase = await decryptFromKeystore(keystore, password)
logger.info('#MARK - Import Wallet Keystore - phrase from keystore', phrase)
```

Severity and Impact Summary

Logging the passphrase leaks private information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.4 Information leakage

Finding ID: KS-XDEFI-F-07

Severity: **Medium**

Status: **Remediated**

Description

Logger leaks information.

Proof of Issue

Filename: config/Background/controllers/WalletsController.ts

Beginning Line Number: 411

```
logger.debug({ wallets })
if (!wallets || !Object.values(wallets).length) {
  return false
}
try {
```

Severity and Impact Summary

Logging the complete wallet could lead to a leak of information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.5 Information leakage

Finding ID: KS-XDEFI-F-11

Severity: **Medium**

Status: **Remediated**

Description

Logger leaks information.

Proof of Issue

Filename: source/containers/Authenicated/Send/BinanceSmartChainSend/index.tsx

Beginning Line Number: 263

```
logger.info('#MARK - test send BSC: wallet password', walletPassword)
```

Severity and Impact Summary

Logging the wallet password could lead to a leak of information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.6 Information leakage

Finding ID: KS-XDEFI-F-13

Severity: **Medium**

Status: **Remediated**

Description

Logger leaks information.

Proof of Issue

Filename: source/containers/UnAuthenticated/MasterKey/index.tsx

Beginning Line Number: 49

```
try {
  const phrase = await masterKeys.createMasterKeystore(data.password)
  setMasterPhrase(phrase)
} catch (error) {
  logger.info('#MARK - MASTER_KEYSTORE_FAILED:', error)
  alert.error(strHCaptialize(error.message || error.toString()))
}
```

Severity and Impact Summary

Logging the Master phrase could lead to a leak of information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.7 Private information leakage

Finding ID: KS-XDEFI-F-16

Severity: **High**

Status: **Remediated**

Description

Debugger leaks private information.

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/bottomActions/index.tsx

Beginning Line Number: 150

```
const seedPhrases = await Promise.all(seedPhrasePromises)
console.log(seedPhrases)
debugger
```

Severity and Impact Summary

The debug statement enables any debugging instrumentation to access the internals of this function. This includes the "seed phrase".

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

4.8 Information leakage

Finding ID: KS-XDEFI-F-23

Severity: **Low**

Status: **Remediated**

Description

Logger leaks information.

Proof of Issue

Filename: source/ext/dapp/transaction/DappTransaction.tsx

Beginning Line Number: 224, 235, 251, 367, 385, 429

```
Logger.info('abi ===> ', inputs)¨

Logger.info('decoded ===> ', decodedParams)

Logger.info('walletId ===> ', walletId)

Logger.info('messages ===> ', msg, txData)

Logger.info('messages ===> ', msg)

Logger.info('reHandleApprove ', selectedMessage)
```

Severity and Impact Summary

Possible information leak through logging of parameters.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information

References

- N/A

5. OTHER OBSERVATIONS

This section contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

5.1 Use of Magic Numbers

Finding ID: KS-XDEFI-O-01

Severity: **Informational**

Description

Refrain from using "Magic Numbers" as these may change or misleading.

Proof of Issue

Filename: source/components/TxPending/index.tsx

Beginning Line Number: 41

```
const safeLowGasPrice = round(Number(safeLow) * 0.1) - 2
```

Severity and Impact Summary

If a “magic number” changes or is misunderstood the calculations in later stages could result in errors that may impact costs for the user in overpayment when performing a transaction.

Recommendation

Introduce a constant that can be defined in an external file to be included in all files needing the constants.

References

N/A

5.2 Missing dependencies

Finding ID: KS-XDEFI-O-02

Severity: **Informational**

Description

React Hook useEffect has missing dependencies: 'averageGasPrice', 'history', 'txId', and 'visibleCancelTx'. Either include them or remove the dependency array.

Proof of Issue

Filename: source/components/TxPending/index.tsx

Beginning Line Number: 107

```
    if (!_transaction && !visibleCancelTx) {
      history.push(`${HOME_URL}?tab=ethereum`)
    }
  }, [pendingTransactions])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verify the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.3 Nonfunctioning method

Finding ID: KS-XDEFI-O-03

Severity: **Informational**

Description

Nonfunctioning method as all code is out commented!

Proof of Issue

Filename: config/Background/controllers/MasterController.ts

Beginning Line Number: 68-69

```
// updater: fecth accounts and balances live from Redux
updater = () => {
  // this.wallets.updateAccounts()
  // setTimeout(() => balances.updateCurrentAccount(), 0)
}
```

Severity and Impact Summary

The functionality to update live from the Redux state is not enabled. This may lead to late updates or wrong data being used in transactions.

Recommendation

Implement the intended functionality or remove the code.

References

- N/A

5.4 Information leakage

Finding ID: KS-XDEFI-O-08

Severity: **Informational**

Description

Logger leaks information

Proof of Issue

Filename: source/containers/Authenicated/Send/BinanceSmartChainSend/index.tsx

Beginning Line Number: 87

```
logger.debug({ assetKey, assets })
```

Severity and Impact Summary

Debug code registering assetkey and assets connected to that key, could lead to a leak of information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

5.5 Use of Magic Numbers

Finding ID: KS-XDEFI-O-09

Severity: **Informational**

Description

Refrain from using "Magic Numbers" as these may change or misleading.

Proof of Issue

Filename: source/containers/Authenicated/Send/BinanceSmartChainSend/index.tsx

Beginning Line Number: 97

```
gasLimit:
  query.get('gasLimit') ||
  String(!isTokenTransfer ? DEFAULT_GAS_LIMIT : DEFAULT_GAS_LIMIT * 3),
```

Severity and Impact Summary

If a “magic number” changes or is misunderstood the calculations in later stages could result in errors that may impact costs for the user in overpayment when performing a transaction.

Recommendation

Introduce a constant that can be defined in an external file to be included in all files needing the constants

References

- N/A

5.6 Missing dependencies

Finding ID: KS-XDEFI-O-10

Severity: **Informational**

Description

React Hook useEffect has missing dependencies: 'chains', 'fromAddress', and 'txData'. Either include them or remove the dependency array.

Proof of Issue

Filename: source/containers/Authenicated/Send/BinanceSmartChainSend/index.tsx

Beginning Line Number: 209

```
useEffect(() => {
  chains
    .getBinanceSmartChainController()
    .getTransactionCount(activeNetwork, fromAddress)
    .then((txCount: any) => {
      logger.debug({ txCount })
      if (!txData.nonce || txData.nonce === '0') {
        setTxData({ ...txData, nonce: txCount.toString() })
      }
    })
}, [activeNetwork])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verifiy the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.7 Missing dependencies

Finding ID: KS-XDEFI-O-12

Severity: **Informational**

Description

React Hook useEffect has missing dependencies: 'balance' and 'totalFee'. Either include them or remove the dependency array.

Proof of Issue

Filename: source/containers/Authenticated/Send/BitcoinSend/index.tsx

Beginning Line Number: 114

```
useEffect(() => {
  const maxAmount = new BigNumber(balance)
    .minus(new BigNumber(totalFee))
    .multipliedBy(1e-8)
    .toNumber()

    if (Number(amount) >= Number(maxAmount)) {
      setAmountExceedsAssetBalance(true)
    } else {
      setAmountExceedsAssetBalance(false)
    }
}, [amount])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verifiy the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.8 Missing dependencies

Finding ID: KS-XDEFI-O-14

Severity: **Informational**

Description

React Hook useEffect has missing dependency: 'isEdit'. Either include it or remove the dependency array.

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/bottomActions/index.tsx

Beginning Line Number: 55

```
useEffect(() => {
  const messageToApprove = isEdit ? selectedTxs : messages
  const accByTx = getTxsByAccounts(messageToApprove, wallets, network)
  const accWithAddress = Object.keys(accByTx).map((walletId) => ({
    walletId,
    label: accByTx[walletId][0].walletLabel,
    seedPhrase: '',
    xdefiIds: accByTx[walletId].map((msg) => msg.xdefiId),
  }))
  setActiveAccounts(accWithAddress)
}, [messages, wallets, network, selectedTxs])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verifiy the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.9 Non implemented crypto assets

Finding ID: KS-XDEFI-O-15

Severity: **Informational**

Description

The code has TODO:s pointing out that the implementation of BITCOIN and BINANCEDEX aren't complete yet and needs to be implemented.

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/bottomActions/index.tsx

Beginning Line Number: 119

```
    }
    case MessageNamespace.BINANCEDEX: // TODO: support
    case MessageNamespace.BITCOIN: // TODO: support
    default:
      throw new Error('Not Supported dApps')
  }
```

Severity and Impact Summary

The functionality of the application may lead to the idea that all crypto assets may be handled. This may lead to unintended consequences if a user can't perform an action.

Recommendation

TODO for handling of BITCOIN and BINANCEDEX not yet implemented. This should be implemented or removed before launch.

References

- N/A

5.10 Potential information disclosure

Finding ID: KS-XDEFI-O-17

Severity: **Informational**

Description

Missing finally clause logs more information than needed.

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/signRequested/index.tsx

Beginning Line Number: 66

```
Logger.debug(`orgMessage: ${orgMessage}`)
try {
  decodedData = JSON.parse(orgMessage)
} catch {
  Logger.debug('orgMessage is not object')
}
Logger.debug(`orgMessage: ${orgMessage}`)
```

Severity and Impact Summary

If not, the logger will double log this specific item (and always log the orgmessage). This could contain sensitive information.

Recommendation

A finally-clause should be used to see to that the code doesn't log any unnecessary information.

References

- N/A

5.11 Missing dependencies

Finding ID: KS-XDEFI-O-18

Severity: **Informational**

Description

React Hook useEffect has missing dependency: 'history'. Either include it or remove the dependency array.

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/signRequested/index.tsx

Beginning Line Number: 85

```
useEffect(() => {
  if (messages.length > 1 && history?.location?.state?.singleElement) {
    history.goBack()
  }
}, [messages])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verifiy the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.12 Password handling unclear

Finding ID: KS-XDEFI-O-19

Severity: **Informational**

Description

TODO for handling Password. Not clear what the TODO refers to.

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/signRequested/index.tsx

Beginning Line Number: 96

```
const checkPwd = async (_pwd = '') => {
    // TODO - handle password
    if (!pwd) {
      alert.show('Please input master password')
      return
    }
```

Severity and Impact Summary

The TODO refers to how the password is handled or not handled. This introduces an uncertainty about what and how the password should be handled.

Recommendation

The unclear reference to Password must be handled. Either remove the comment or be more clear in stating the intention. See to that the intended functionality is implemented.

References

- N/A

5.13 Information leakage

Finding ID: KS-XDEFI-O-20

Severity: **Informational**

Description

Debugger leaks information

Proof of Issue

Filename: source/ext/dapp/multiTxMsgSign/components/signRequested/index.tsx

Beginning Line Number: 162

```
parsed = JSON.parse(orgMessage)
Logger.debug({ parsed })
```

Severity and Impact Summary

Debug logging of the orgMessage could lead to a leak of information that may be picked up by an adversary.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information

References

- N/A

5.14 Missing dependencies

Finding ID: KS-XDEFI-O-21

Severity: **Informational**

Description

React Hook useEffect has missing dependency: 'history'. Either include it or remove the dependency array.

Proof of Issue

Filename: source/ext/dapp/transaction/DappTransaction.tsx

Beginning Line Number: 90

```
useEffect(() => {
  if (messages.length > 1 && history.location?.state?.singleElement) {
    history.goBack()
  }
}, [messages])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verifiy the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.15 Information leakage

Finding ID: KS-XDEFI-O-22

Severity: **Informational**

Description

Debugger leaks information.

Proof of Issue

Filename: source/ext/dapp/transaction/DappTransaction.tsx

Beginning Line Number: 196

```
console.log('RES ==> ', res)
if (res !== -1) {
  return Object.values(wallets)[res].id
}
return null
```

Severity and Impact Summary

: Debug code still active: console.log dumps result value to the Javascript Console. This could create a information leak.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

5.16 Missing dependencies

Finding ID: KS-XDEFI-O-24

Severity: **Informational**

Description

React Hook useEffect has missing dependencies: 'controller.fees', 'decryptParams', 'namespace', 'raw.params', and 'txData'. Either include them or remove the dependency array.

Proof of Issue

Filename: source/ext/dapp/transaction/DappTransaction.tsx

Beginning Line Number: 314

```
    } else if (namespace === MessageNamespace.BINANCESMARTCHAIN) {
      setChainId(IChainType.binancesmartchain)
      setSymbol('BNB')
      setSigInfo(decryptParams())
      setChainIcon(BinanceIcon.toString())

      setTxData({
        ...txData,
        gasPrice: String(round(controller.fees.getBscGasPrice().average)),
      })
    }
  }, [signature])
```

Severity and Impact Summary

The risk of having dependencies not being updated while invoking a react hook could lead to stale closures with unintended consequences when it comes to data. This may lead to the wrong data being used in situations where is would be crucial not to.

Recommendation

Verifiy the list of dependencies for Hooks like useEffect and similar, protecting against the stale closure pitfalls. See to defining all dependencies to be explicitly clear.

References

- https://dmitripavlutin.com/react-hooks-stale-closures/

5.17 Information leakage

Finding ID: KS-XDEFI-O-25

Severity: **Informational**

Description

Debugger leaks information

Proof of Issue

Filename: containers/Unauthenticated/ImportWallet/StandardImport.tsx

Beginning Line Number: 94

```
        logger.info(e.target?.result)
```

Severity and Impact Summary

Debug code still active: dumping the whole file read from storage into the log. This could create a information leak.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information

References

- N/A

5.18 Information leakage

Finding ID: KS-XDEFI-O-26

Severity: **Informational**

Description

Debugger leaks information

Proof of Issue

Filename: containers/Unauthenticated/ImportWallet/StandardImport.tsx

Beginning Line Number: 100

```
        } catch (err) {
          console.error(err)
          alert.show('Could not import this keystore')
        }
```

Severity and Impact Summary

Debug code still active: console.log dumps result value to the Javascript Console. This could create a information leak.

Recommendation

Do not log private information at any stage. If the information needs to be inspected, find a way to encrypt the information before storage so that only the intended recipient will be able to analyze the information.

References

- N/A

APPENDIX A: ABOUT KUDELSKI SECURITY

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security

route de Genève, 22-24

1033 Cheseaux-sur-Lausanne

Switzerland

Kudelski Security

5090 North 40th Street

Suite 450

Phoenix, Arizona 85018

APPENDIX B: DOCUMENT HISTORY

VERSION	STATUS	DATE	AUTHOR	COMMENTS
0.1	Draft	19 March 2021	Mikael Björn	
1.0	Final	1 April 2021	Mikael Björn	

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Head of Research	19 March 2021	0.1	
Nathan Hamiel	Head of Research	2 April 2021	1.0	

APPENDIX C: SEVERITY RATING DEFINITIONS

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, providing customers with a quick at a glance view of the risk an issue poses to the system. As with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

SEVERITY	DEFINITION
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.