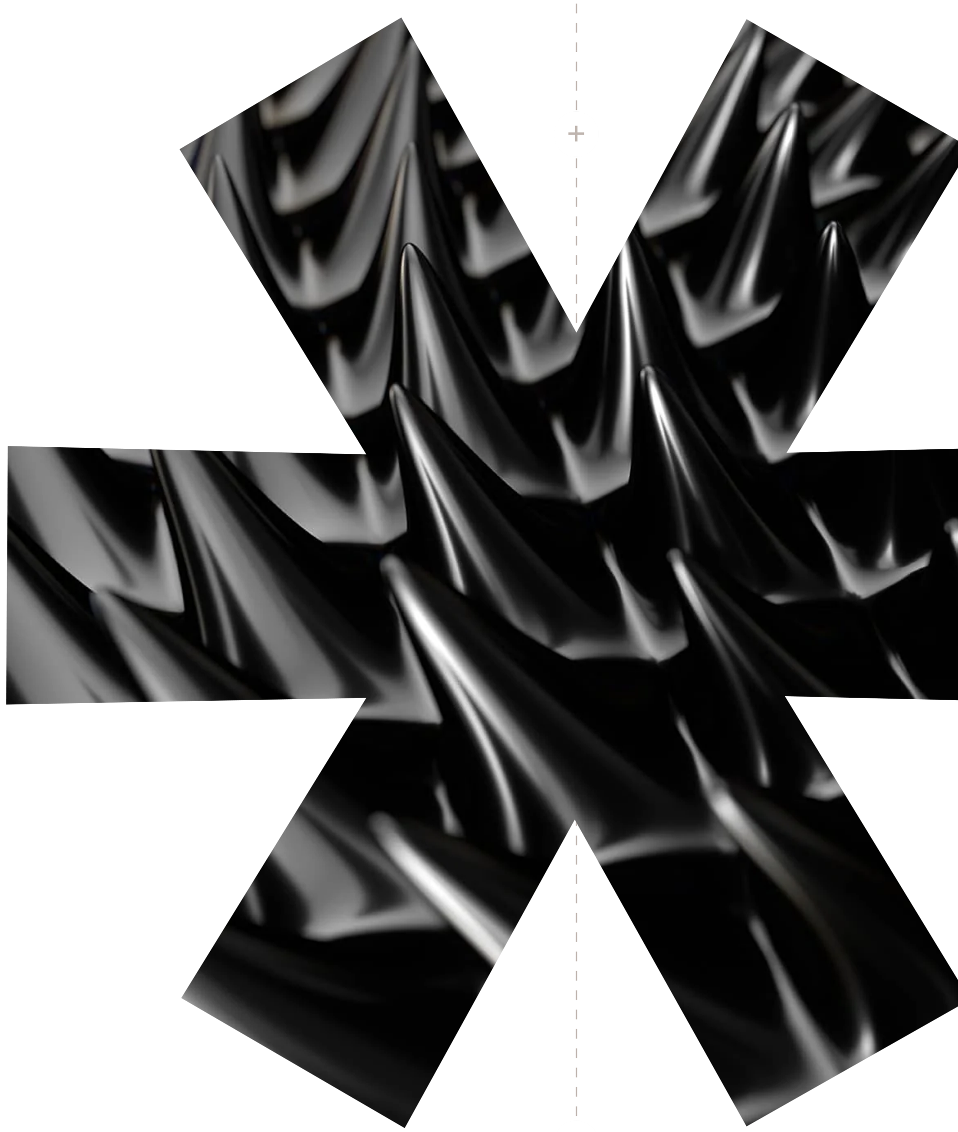
thesisdefense

Security Audit Report

Mezo

Mezo Smart Contracts

Initial Report // April 5, 2024
Final Report // April 19, 2024



Team Members

Bernd Artmüller // Security Auditor
J4X // Security Auditor
Bashir Abu-Amr // Head of Delivery

Table of Contents

About Thesis Defense

Thesis Defense serves as the auditing services arm within Thesis, Inc., the venture studio behind tBTC, Fold, Taho, Etcher, and Mezo. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of technologies including smart contracts, wallets and browser extensions, bridges, node implementations, cryptographic protocols, and dApps. We offer our services within a variety of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos / Cosmos SDK, NEAR and more.

Thesis Defense will employ the Thesis Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

Section_1.0

Scope

Technical Scope

- **Repository:** https://github.com/thesis/orangekit/tree/main/solidity
- **Audit Commit:** 44355ad8dbac7df34d069fca2720c9e3f96b0ff9

Documentation

- Technical documentation and architectural diagram:RFC: OrangeKit Bitcoin Account Metaprotocol
- The instructions for setting up the repository and running tests are available in the README file

Reference:

- Message signing - Bitcoin Wiki. (n.d.)

Section_2.0

Executive Summary

Schedule

This security audit was conducted from April 1, 2024 to April 5, 2024 by 2 security auditors for a total of 2 person-weeks.

Overview

Thesis Defense conducted a manual code review of Mezo OrangeKit smart contract implementation.

The Mezo OrangeKit protocol allows users to interact with the Ethereum EVM ecosystem using their Bitcoin private key. This is facilitated by using an ERC-4337 compatible Gnosis Safe and a custom implementation of a Safe owner. This Safe owner smart contract can verify different Bitcoin signatures and allows the user to control the actions of the Gnosis Safe by signing messages and sending them to the Safe to execute.

Threat Model

For this review, our team considered a threat model whereby external components to the smart contracts are untrusted but function as intended. These components include any user interface that enables interaction with the protocol, any off-chain components that are an integral part of the system, and any third-party dependencies or services that are necessary for the protocol to function as intended. Furthermore, we considered the governance of the protocol to be not malicious. Due to the usage of the battle-tested Gnosis Safe, the attack surface of the OrangeKit contracts is very limited. Nevertheless, multiple threats were considered in the audit:

- Forgeability of signatures
- Replaying of signatures
- Unauthorized access to the safe / deployer / factory
- Unpredictability of addresses

The main attackers considered are attackers trying to pass invalid or already used signatures to get the Gnosis Safe to act maliciously.

Security by Design

The OrangeKit system design is robust, and the protocol's security has been considered and prioritized. The usage of the Gnosis Safe contract instead of a custom implementation also supports this fact. Throughout the code, authorization is correctly implemented, and many potential attack vectors are already mitigated through various security measures. For instance, replaying transactions is prevented by the Gnosis Safe contract by incorporating the `chainid` as well as an increasing nonce in the message digest. The additional possibility of emergency upgrades through a trusted entity allows the governance (Multisig) to take remediative action in case of a compromised implementation of the Safe owner smart contract.

Secure Implementation

We found the code to be well-organized, properly documented, and adhering to best practices. We investigated the security of the implementation of the two most sensitive areas: signature verification and contract deployment to predictable addresses. As a result of our review, we did not identify any critical security vulnerabilities.

Use of Dependencies

We ran the pnpm audit dependency analysis tool and did not identify any issues in the use of Solidity dependencies.

Tests

The OrangeKit repository contains unit and integration tests for the contracts in the scope of this review, with 96.79% line and 93.52% branch coverage, in accordance with best practices.

We recommend adding tests for the OrangekitDeployer contract that try to redeploy the individual contracts, which results in an error because they use the same salt and bytecode.

Project Documentation

The OrangeKit smart contracts are well documented in RFC: OrangeKit Bitcoin Account Metaprotocol, provided by the client. This documentation encompasses detailed technical descriptions of the smart contracts' essential functions and is also supplemented by an architectural diagram. Moreover, the code itself is well-commented.

In addition, we recommend making the available documentation publicly available to make it easier for developers to inform themselves about the workings of the protocol.

Section_3.0

Key Findings Table

Issues	Severity	Status
ISSUE #1 The `BitcoinSafeOwner` and `OrangeKitSafeFactory` Contracts can be Upgraded with a Non-Contract Address Resulting in a Non-Functional Proxy	Low	Not Fixed
ISSUE #2 Two-Step Ownership Transfer is Recommended	Low	Not Fixed
ISSUE #3 Rename the `EmergencyUpgradesDisabled` Error	None	Not Fixed
ISSUE #4 Private and Internal Functions Do Not Adhere to the Solidity Style Guide	None	Not Fixed
ISSUE #5 Comment Includes Spelling Issue	None	Not Fixed
ISSUE #6 Missing NatSpec Return Value Documentation	None	Not Fixed
ISSUE #7 Add Indexing to Events With Multiple Parameters	None	Not Fixed
ISSUE #8 Missing Derived Address Prefix Allows Users to Pass Compressed P2PKH Signatures as P2WPKH Signatures	None	Not Fixed
ISSUE #9 Emit an Additional Event During Emergency Upgrades of the `BitcoinSafeOwner` Contract	None	Not Fixed
ISSUE #10 Avoid Using the Latest Solidity Compiler Version to Reduce Compiler Bug Risks	None	Not Fixed
ISSUE #11 No Protection Against Outdated Version Upgrade	None	Not Fixed

Severity definitions can be found in Appendix A

Section_4.0

Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

The `BitcoinSafeOwner` and `OrangeKitSafeFactory` Contracts can be Upgraded with a Non-Contract Address Resulting in a Non-Functional Proxy

Low | Not Fixed

Location

BitcoinSafeOwner.sol#L157-L162

OrangeKitSafeFactory.sol#L171-L176

Description

The `BitcoinSafeOwner` and `OrangeKitSafeFactory` contracts can be upgraded via the `upgradeSingleton` function by changing the proxy's implementation address, stored in singleton. The address of the new implementation contract is validated to prevent the zero address, the proxy's address, or the address of the current implementation contract from being set as the new implementation.

However, if the address provided does not refer to a contract, the upgrade will succeed due to the proxy's low-level `delegatecall` not reverting if the target is not a contract. As a result, the proxy will be non-functioning and cannot be upgraded again as the upgrade functionality is not available.

Impact

The upgraded smart contracts are non-functional and cannot be upgraded again.

Recommendation

We recommend using the `isContract` function to check if the new implementation address is a contract before proceeding with the upgrade.

Verification Status

The Mezo team has stated that a restrictive process for upgrades is in place such that for every upgrade, there is a checklist reviewed by at least two developers, with all transactions simulated and reviewed in Tenderly, and a required minimum number of technical people to sign a transaction. However, at the time of this verification, the suggested remediation has not been implemented.

ISSUE#2

Two-Step Ownership Transfer is Recommended

Low | Not Fixed

Location

OrangeKitSafeFactory.sol#L192-L199

Description

The `OrangeKitSafeFactory` implements an ownership mechanism that is used to allow the `EmergencyUpgrader` to upgrade the singletons in case of a potential compromise. The functionality currently only allows for a single-step ownership transfer, which could lead to issues if the ownership is accidentally transferred to the wrong address.

Impact

If the ownership of the `OrangeKitSafeFactory` is transferred to an unintended address by accident it will make it impossible for the `EmergencyUpgrader` to upgrade the singletons in the case of the used one becoming vulnerable. This will result in new safes being deployed with a vulnerable version without a way of preventing it.

Recommendation

We recommend implementing a two-step ownership transfer. This can be done by using an already implemented library implementation like OpenZeppelin. To still grant the ownership to the `EmergencyUpgrader` when calling `initialize`, the function can be adapted so that an owner parameter can be passed that is set when initializing.

Verification Status

The Mezo team has stated that a restrictive process for upgrades is in place such that for every upgrade, there is a checklist reviewed by at least two developers, with all transactions simulated and reviewed in Tenderly, and a required minimum number of technical people to sign a transaction. However, at the time of this verification, the suggested remediation has not been implemented.

ISSUE#3

Rename the `EmergencyUpgradesDisabled` Error

None | Not Fixed

Location

EmergencyGovernance.sol#L28

Description

The OrangeKit protocol implements an emergency governance feature that allows a MultiSig controlled by the protocol itself to upgrade the `BitocinSafeOwner`. To inform the deployed `BitcoinSafeOwner` smart contracts what address currently holds the `EmergencyUpgrader` role, the `EmergencyGovernance` smart contract is used. This smart contract can also be disabled by its owner, which will make emergency upgrades impossible afterward. After the `EmergencyGovernance` was disabled, the `EmergencyUpgradesDisabled` error is thrown at each call to its 3 functionalities:

1. Retrieving the current emergency upgrader

2. Disabling the `EmergencyGovernance`
3. Setting a new emergency upgrader

As the error's name suggests, it is intended to be returned when the emergency governor tries to upgrade a safe owner, but the `EmergencyGovernance` is already disabled. In the other two cases, the error might lead to confusion.

Impact

None – no security impact.

Recommendation

We recommend splitting the error into three separate errors (which would increase overhead) or renaming it to a more generic version that fits all three cases of reverting. One recommendation is to rename the error to `EmergencyGovernanceDisabled`.

Verification Status

The Mezo team deployed the smart contracts without implementing the recommendation and stated that they intend to apply them in the future.

ISSUE#4

Private and Internal Functions Do Not Adhere to the Solidity Style Guide

None | Not Fixed

Location

OrangeKitSafeFactory.sol#L208, L270, L322, L339, L359, L396

OrangeKitDeployer.sol#L137

BitcoinSafeOwner.sol#L278, L371, L398, L425, L456, L480, L492, L516, L539

Description

To ensure good code readability and prevent future issues, it is highly recommended that Solidity code follow the Solidity Style Guide. The style guide states that non-external functions should be prefixed with an underline. This currently needs to be implemented for many functions used in the protocol.

Impact

None – no security impact.

Recommendation

We recommend prefixing the private and internal functions accordingly.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#5

Comment Includes Spelling Issue

None | Not Fixed

Location

BitcoinSafeOwner.sol#L317

Description

One of the comments inside the `BitcoinSafeOwner` includes a typo in the word "varint" which should correctly be "variant."

Impact

None – no security impact.

Recommendation

We recommend fixing the spelling issue.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#6

Missing NatSpec Return Value Documentation

None | Not Fixed

Location

BitcoinSafeOwner.sol#L127

EmergencyGovernance.sol#L44

LegacyERC1271.sol#L34

OrangeKitSafeFactory.sol#L99, L122, L208, L270

Description

It is recommended to use NatSpec documentation to improve code readability. Throughout the codebase, the @return tag is used most of the time correctly to document the returned variables. Still, some functions are fully missing the comment or have a comment describing the returned value but with an incorrect tag.

Impact

None – no security impact.

Recommendation

We recommend documenting the return value in NatSpec for each of the functions.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#7

Add Indexing to Events With Multiple Parameters

None | Not Fixed

Location

OrangeKitSafeFactory.sol#L39-L40

BitcoinSafeOwner.sol#L58

EmergencyGovernance.sol#L19

Description

The OrangeKit protocol correctly emits events on state changes. To make it easier to monitor these events it is recommended to index them if they have more than one argument. This is correctly implemented for one event but not for all.

Impact

None – no security impact.

Recommendation

We recommend indexing the mentioned events to improve monitoring abilities.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#8

Missing Derived Address Prefix Allows Users to Pass Compressed P2PKH Signatures as P2WPKH Signatures

None | Not Fixed

Location

BitcoinSafeOwner.sol#L456

Description

The OrangeKit protocol implements the `BitcoinSafeOwner` contract which can be used to verify messages signed by a Bitcoin address. These messages can be verified in four ways depending on the type of address that has encoded them. For the address types `compressedP2PKH` and `P2WPKH` the signatures are verified the same way. The only way the contract distinguishes between them is by checking the `v` value and then decreasing it by 8 if `P2WPKH` is detected.

```
 uint8 prefix = uint8(uint256(y) & 1) + uint8(2);
bytes20 publicKeyHash = hash160(abi.encodePacked(prefix, x));

return
  ecrecover(signedMessage, v, r, s) ==
  publicKeyToEthereumAddress(x, y) &&
  truncatedBitcoinAddress == publicKeyHash;
```

Consequently, any compressed `P2PKH` signature can also be passed as a `P2WPKH` signature by increasing the value of `v` by 8.

Fortunately, this currently does not lead to issues as the Gnosis Safe protects against message replays by using nonces inside the messages. Nevertheless, this could lead to issues if the `BitcoinSafeOwner` is ever used with a different multi-sig contract, that, for example, protects against replay attacks by creating a digest of the message + signature. In that case, the message could be used once as compressed `P2PKH` and once as `P2WPKH`.

Impact

None – no security impact.

Recommendation

We recommend adding an enum for the type of address used when creating the Safe and only allowing signature verification of that type.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#9

Emit an Additional Event During Emergency Upgrades of the `BitcoinSafeOwner` Contract

None | Not Fixed

Location

BitcoinSafeOwner.sol#L215

Description

In case of emergency, the `BitcoinSafeOwner` contract can be upgraded by the governance upgrader via the `emergencyUpgradeSingleton` function. As a result, the event `SingletonUpgraded(address oldSingleton, address newSingleton)` is emitted, similarly to a regular upgrade by the contract owner via the `upgradeSingleton` function. Emitting an additional event in case of an emergency upgrade would allow better differentiation between governance and owner upgrades for more effective off-chain monitoring.

Impact

None – no security impact.

Recommendation

We recommend emitting an additional event, for example, `SingletonUpgradedEmergency`, in the `emergencyUpgradeSingleton` function.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#10

Avoid Using the Latest Solidity Compiler Version to Reduce Compiler Bug Risks

None | Not Fixed

Location

BitcoinSafeOwner.sol#L2

ERC1271.sol#L2

EmergencyGovernance.sol#L2

LegacyERC1271.sol#L2

OrangeKitDeployer.sol#L2

OrangeKitSafeFactory.sol#L2

Proxy.sol#L2

Description

All smart contracts in the scope of this review use the `pragma solidity 0.8.25` statement. This sets the version of the Solidity compiler, solc, to 0.8.25, which is the latest version at the time of this review. However, new compiler versions can occasionally introduce bugs and unknown vulnerabilities, and therefore, using the latest compiler version may pose a risk.

Impact

None – no security impact.

Recommendation

We recommend not using the latest Solidity compiler version, especially if none of the latest compiler features are used. This reduces the risk of potentially introducing unknown compiler bugs. For example, consider using solc version 0.8.24.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

ISSUE#11

No Protection Against Outdated Version Upgrade

None   Not Fixed

Location

BitcoinSafeOwner.sol#L152

Description

The `BitcoinSafeOwner` smart contract allows upgrades to include new functionalities or remove vulnerabilities. To protect the upgrading process against signature replay attacks, i.e., redeploying previous contract versions, each message includes the domain separator that includes the version number. As long as the version number increases at each upgrade, the signature can never be replayed.

Unfortunately, besides code comments pointing out the necessity to strictly increase the version, there are no explicit checks in the code that enforce this. The code only checks that the new initializer function is called with a function value, but no check is done to verify that the version number has increased.

Impact

None – no security impact.

Recommendation

There are two ways to mitigate this issue:

- **Hash mapping:** The more gas-intensive way would be to implement an additional mapping of `keccak256(versionString) => bool`. At the end of the `setup` function, it must be checked that the hash of the new version is not already contained in that mapping, reverting otherwise. If the version is indeed new, it is added to the mapping, and the upgrade succeeds. This way, the version formatted as a string can stay, but this comes at the cost of increased gas costs.
- **Numerical version number:** The less gas-intensive way of securing users against accidentally upgrading to an old version is to use numerical version numbers. This can be done by replacing the version number string with an `uint256`. In this case, the function can easily check if the new version number is higher than the old one and revert otherwise.

Verification Status

The Mezo team deployed the smart contracts without implementing the issues identified in this report and stated that they intend to address them in the future.

Appendix A

Severity Rating Definitions

At Thesis Defense, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Severity	Definition
Critical	• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties • Permanent freezing of funds • Permanent freezing of NFTs • Unauthorized minting of NFTs • Predictable or manipulable RNG that results in abuse of the principal or NFT • Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content) • Protocol insolvency
High	• Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds • Temporary freezing NFTs
Medium	• Smart contract unable to operate due to lack of token funds • Enabling/disabling notifications • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption
Low	• Contract fails to deliver promised returns, but doesn't lose value
None	• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.

Section_6.0

Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.