CLOUDWALK MULTISIG WALLET SECURITY AUDIT REPORT

April 27, 2023

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The purpose of multi-sig wallets is to improve security by requiring multiple parties to agree on transactions before execution. The transactions can only be executed when confirmed by a predefined number of owners.

1.4 Project Dashboard

Project Summary

Title	Description
Client	CloudWalk
Project name	MultiSig Wallet
Timeline	March 14 2023 - April 19 2023
Number of Auditors	3

Project Log

Date	Commit Hash	Note
14.03.2023	b5d6c2b6273162d5666d48649890b15a113df7a7	Commit for the audit

Date	Commit Hash	Note
24.03.2023	91605cfd0a3f99faed1ba2c85deb1c8ea1f93438	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
MultiSigWalletFactory.sol	MultiSigWalletFactory.sol
MultiSigWallet.sol	MultiSigWallet.sol
MultiSigWalletUpgradeable.sol	MultiSigWalletUpgradeable.sol
MultiSigWalletBase.sol	MultiSigWalletBase.sol

Deployments

Network: Ethereum

Contract	Address	Creation TX hash
MultiSigWalletFactory	0x39977806d4bfb71c3e7ff8c4226d3b13ee38ad1b	0x440e9bb88a1b2a0eb1f8aa49a97645d9c1d49a6824546ed430074851665ccf74

Network: CloudWalk

Contract	Address	Creation TX hash
MultiSigWalletFactory	0x0e4CEF226e0808c30957F690E75E076477AAEaE3	0x9020a1f9b7af493a2e96a3a5027cced9b15520bb9229879dac48201541929589

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	2
Medium	1
Low	4

ID	Name	Severity	Status
C-1	Double-approval attack by malicious owner	Critical	Acknowledged
H-1	`_execute` allows you to execute unsuccessful tasks in the future	High	Acknowledged
H-2	A transaction doesn't have the sequence information	High	Acknowledged
M-1	`WalletUpgradable` initialization is risky	Medium	Fixed
L-1	`Wallet._configureExpirationTime()` allows setting low values, it's likely for the transactions to be frozen	Low	Fixed
L-2	Transfer ownership in two steps	Low	Acknowledged
L-3	Transaction exposure	Low	Acknowledged
L-4	Inability to cancel transactions with long expiration times after owner replacement	Low	Acknowledged

1.6 Conclusion

During the audit process, 1 critical, 2 high, 1 medium, and 4 low severity findings were spotted and confirmed by the developers. Having worked on the reported findings, all of them were acknowledged or fixed by the client. Critical and high vulnerabilities remained unfixed, but they can be acknowledged without severe impact on the project security if only the protocol users are aware of how the contract works. The protocol team confirmed that this information is contained in the Cloudwalk's documentation and all users know where to find it.

2. FINDINGS REPORT

2.1 Critical

C-1	Double-approval attack by malicious owner
Severity	Critical
Status	Acknowledged

Description

Pending transactions have the `_approvalCount` counter.

The `_approvalCount` for each transaction is simply incremented by 1 each time an owner approves it. However, when the owners are reconfigured using the `configureOwners()` function, the `_approvalCount` value for valid transactions remains positive. This can lead to a scenario where the minority of owners can execute a transaction.

- MultiSigWalletBase.sol#L345
- MultiSigWalletBase.sol#L370

For example, in a wallet requiring the 2/3 of approvals, a malicious owner can create a transaction to withdraw all funds and approve it. They can then claim their account has been hacked and ask the other owners to change their account address (using `configureOwners()`). After the address change, the malicious owner can approve the previously created transaction again reaching the minimum required approvals=2 and execute the transaction.

Recommendation

We recommend removing the counter and checking approvals in `_execute()` by iterating through `_approvalStatus()` using current owners.

Like here in Gnosis:

- MultiSigWallet.sol#L267-L279.

Client's commentary

> This is expected behavior. Wallet owners must understand how a multisig contract works and what they do when interacting with it. Also, there is always an option to revoke your past approvals before submitting new wallet owners. We have expanded the multisig wallet documentation to explain all the side effects of changing wallet owners.
> MB: If multisig owners know about the issue and this will be taken into account in the documentation, then the status can be Acknowledged without severe impact on the protocol security.

2.2 High

H-1	`_execute` allows you to execute unsuccessful tasks in the future
Severity	High
Status	Acknowledged

Description

If the transaction is unsuccessful, then a revert will occur in the code (MultiSigWalletBase.sol#L378):

```
(bool success, bytes memory data) = transaction.to.call{
    value: transaction.value
}(transaction.data);
if (!success) {
    revert InternalTransactionFailed(data);
}
```

However, if a condition allows the transaction to be executed later, it will be completed as success. This is especially dangerous for:

- `expirationTime` as `365 days`
- various swaps with a slippage
- `configureOwners` method and others.

Recommendation

We recommend disallowing to use the transaction after `revert`, because the permission to execute remains with this transaction.

Client's commentary

> This is expected behavior. Wallet owners must understand how a multisig contract works and what they do when interacting with it. Also, there is always an option to revoke your past approvals if you don't want the transaction to be executed in the future. We have expanded the multisig wallet documentation to explain what happens if the internal call is unsuccessful.
> MB: If the owners know about the issue and this will be taken into account in the documentation, then the status is Acknowledged.

H-2	A transaction doesn't have the sequence information
Severity	High
Status	Acknowledged

Description

In case of multiple transactions are approved any owner is able to choose the sequense of transactions. In some cases this owner can extract additional benefits from choosing the ordering.

In addition, transactions allow only one target call per transaction - it may be not enough for DeFi interactions. Complex DeFi interactions require making a few approved transactions, so benefits from sequencing can arise.

- IMultiSigWallet.sol#L11-L18
- MultiSigWalletBase.sol#L378

Recommendation

We recommend adding the nonce information in transactions and allowing a list of targets.

Client's commentary

> This is expected behavior. Wallet owners must understand how a multisig contract works and what they do when interacting with it. Also, there is always an option to revoke your past approvals if you want to change the order of transaction execution. We have expanded the multisig wallet documentation to explain that approved transactions can be executed in any order.
> MB: If the owners know about the issue and this will be taken into account in the documentation, then the status is Acknowledged.

2.3 Medium

M-1	`WalletUpgradable` initialization is risky
Severity	Medium
Status	Fixed in 91605cfd

Description

The current initialization of an upgradable wallet includes multiple steps, where a wallet proxy is owned by some external admin. The admin has to transfer proxy ownership directly to the wallet to finalize the initialization.

Recommendation

We recommend applying the same factory-pattern for wallets as well in order to guarantee correct finalized initialization.

Also one of the solutions is to use `UUPSUpgradeable` with the correct setting `_authorizeUpgrade()` method via `onlySelfCall`.

Client's commentary

> We switched to the UUPS pattern instead of Transparent.

2.4 Low

L-1	`Wallet._configureExpirationTime()` allows setting low values, it's likely for the transactions to be frozen
Severity	Low
Status	Fixed in 91605cfd

Description

`_expirationTime` should be long enough to make a transaction.
If it is low or mistakenly set to zero - transactions will be hard to execute.

- MultiSigWalletBase.sol#L314

Recommendation

We recommend setting a range for `_expirationTime` with minimum and maximum amounts.

Client's commentary

> We introduced a minimum expiration time.
> MB: Only the minimum expiration time is checked now. The maximum exp time is not checked, in edge cases it means that a transaction will not expire.

L-2	Transfer ownership in two steps
Severity	Low
Status	Acknowledged

Description

In the current implementation, the `configureOwners()` function immediately replaces the old owners with the new ones. However, this can lead to a permanent contract lock if it turns out that not all new owners actually control the provided addresses. A safer approach would be to implement a two-step process for changing owners, where the final owner replacement only occurs after all new owners have confirmed their ownership of the specified addresses.

- MultiSigWalletBase.sol#L190

Recommendation

It is recommended to implement a two-step owner replacement process. Instead of immediately replacing the old owners, the function should first require new owners to confirm their ownership of the specified addresses. Only after all new owners have successfully confirmed their ownership, should the final owner replacement take place. This approach reduces the risk of a contract lock due to incorrect or inaccessible owner addresses.

Client's commentary

> This is expected behavior. Wallet owners must understand how a multisig contract works and what they do when interacting with it. Also, there is always an option to revoke your past approvals before submitting new wallet owners. We have expanded the multisig wallet documentation to explain all the side effects of changing wallet owners.

L-3	Transaction exposure
Severity	Low
Status	Acknowledged

Description

In the current implementation, even when `_cooldownTime` is set to 0, users are unable to collect all required approvals and submit a transaction to the private mempool at once. As a result, large trading transactions are visible in the contract state in advance, since the owners must first submit the full transaction data and then approve it through separate transactions. This makes these transactions vulnerable to griefing or profit reduction, as other parties can extract profit by observing the pending transactions and acting accordingly.

- MultiSigWalletBase.sol#L316

Recommendation

Consider implementing a hidden transaction approach, where instead of storing the full transaction data in the contract storage, only the hash of the transaction is stored. This way, the transaction details remain hidden until the transaction is executed. Once the required number of owners have provided valid signatures for the hashed transaction, the transaction can be executed with the actual transaction data provided during execution. This approach adds an extra layer of privacy and security for large trading transactions.

Client's commentary

> This is expected behavior. Wallet owners must understand how a multisig contract works and what they do when interacting with it. We have expanded the multisig wallet documentation to explain that transaction data is publicly visible on the blockchain.

L-4	Inability to cancel transactions with long expiration times after owner replacement
Severity	Low
Status	Acknowledged

Description

In the current implementation, when owners are replaced, transactions with the previously set long `_expirationTime` values continue to remain in the list of pending transactions. New owners are unable to cancel these transactions, potentially allowing malicious transactions to be executed in the future. This poses a security risk, as new owners may not have the desired control over pending transactions submitted by previous owners.

- MultiSigWalletBase.sol#L314

Note, that the `revoke()` function cannot help in that case, because it only revokes a single approval of a specific owner and cannot revoke old approvals if the owners are changed.

- MultiSigWalletBase.sol#L387

Recommendation

To address this vulnerability, implement a mechanism to either automatically cancel or invalidate all pending transactions when owners are replaced. Alternatively, allow new owners to cancel or reject transactions, providing them with the necessary control over the contract's pending transactions. This will help prevent the execution of unwanted or malicious transactions submitted by the previous owners and ensure that only transactions approved by the current set of owners are executed.

Client's commentary

> This is expected behavior. Wallet owners must understand how a multisig contract works and what they do when interacting with it. Also, there is always an option to revoke your past approvals before submitting new wallet owners. We have expanded the multisig wallet documentation to explain all the side effects of changing wallet owners.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts


https://github.com/mixbytes/audits_public


https://mixbytes.io/


hello@mixbytes.io


https://twitter.com/mixbytes