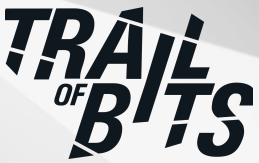

Equilibrium Multisig Contract

Security Assessment

November 1, 2022

Prepared for:
Lauri Peltonen
Equilibrium

Prepared by: **Jim Miller, Gustavo Grieco, Filipe Casal, and Marc Ilunga**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Trail of Bits was contracted to review the security of Equilibrium's StarkNet `Multisig` contract. From September 6 to September 16, 2022, a team of four consultants conducted a security review of the client-provided source code. Details of the project's test targets and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the target system, including access to the source code and documentation. We performed static testing of the target system, using both automated and manual processes.

Summary of Findings

A summary of the findings and details on a notable finding are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
Medium	1
Low	1
Informational	2

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	3
Patching	1

Notable Findings

A significant flaw that impacts system confidentiality, integrity, or availability is described below.

- **TOB-EQMS-1**
  Insufficient input validation in the `Multisig` contract could allow the threshold to be set to zero, which would cause any previously submitted transaction to be automatically approved.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Cara Pearson, Project Manager
cara.pearson@trailofbits.com

The following engineers were associated with this project:

Jim Miller, Consultant
james.miller@trailofbits.com

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Filipe Casal, Consultant
filipe.casal@trailofbits.com

Marc Ilunga, Consultant
marc.ilunga@trailofbits.com

Project Goals

The engagement was scoped to provide a security assessment of Equilibrium's StarkNet `Multisig` contract. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are there any security issues related to the `Multisig` contract's use of Cairo?
- Are access controls for critical operations implemented in the `Multisig` contract?
- Could Amarna be used to identify known Cairo security issues in the `Multisig` contract?

Project Targets

The engagement involved a review and testing of the following target.

StarkNet Multisig

Repository	https://github.com/eqlabs/starknet-multisig
Version	`3c16c8b00ec1706ffa721b5d17bc6642834fb680`
Type	Cairo
Platform	StarkNet

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **`Multisig`.** We manually reviewed the `Multisig` contract; our primary focus was to ensure that the contract uses Cairo safely, has sufficient access controls, and is consistent logically. In addition to this manual review, we used Amarna to identify potential security issues.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	The set_signers function allows a wallet's threshold to be set to zero	Data Validation	**Medium**
2	Multisig constructor and associated functions allow signers to be zero	Data Validation	**Low**
3	Multisig contract does not use the latest version of Cairo	Patching	**Informational**
4	require_tx_exists accepts negative nonces	Data Validation	**Informational**

Detailed Findings

1. The set_signers function allows a wallet's threshold to be set to zero

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EQMS-1
Target: `starknet-multisig/contracts/Multisig.cairo`	

Description

When a multisignature wallet is set up, the wallet's signers and threshold (the number of signers needed to approve a transaction) are specified. The `Multisig` contract includes the `require_valid_threshold` function, which takes as input the threshold and the number of signers. This function ensures that the threshold is at least one (to prevent a threshold of zero, which would allow anyone to execute submitted transactions). This function is used in various locations throughout the `Multisig` contract, such as the constructor and functions like `set_threshold`, which is used to update a wallet's threshold.

The `Multisig` contract also includes the `set_signers` function, which is used to update a wallet's signers. This function does not allow the wallet's threshold to be changed. However, if the number of signers is changed by this function, then the threshold may have to be updated. For instance, if the new number of signers is now less than the existing threshold, then the threshold must be decreased to prevent the wallet from being locked. In the `Multisig` contract, the `set_signers` function accounts for this by automatically changing the threshold to be equal to the new number of signers if the new number is less than the threshold. However, as shown in figure 1.1, the function does not verify that the number of signers is greater than zero; therefore, if a transaction is submitted to change the signers to an empty set, this function will set the wallet's threshold to zero.

```
# @dev Sets a new array of signers. The only way this can be invoked
# is via a recursive call from execute_transaction -> set_signers.
# Threshold is automatically decreased if it's larger than the new number of
signers.
# @param signers_len: New amount of signers
# @param signers: New array of signers
@external
func set_signers{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*, range_check_ptr}(
    signers_len : felt, signers : felt*
):
    alloc_locals
```

```
    require_multisig()

    _set_signers(signers_len, signers)

    let (local threshold) = _threshold.read()
    let (lt) = is_le(signers_len, threshold - 1)  # signers_len < threshold
    if lt == TRUE:
        _set_threshold(signers_len)
        return ()
    end

    return ()
end
```

Figure 1.1: starknet-multisig/contracts/Multisig.cairo#L528-L550

Once the wallet's threshold is set to zero, any submitted transaction can be executed, as no confirmations are needed. Therefore, a malicious signer could submit several malicious transactions. Then, if the attacker is able to trick the other signers of the wallet into approving a transaction that changes the signers of the wallet to the empty set, he will be able to execute all of his malicious transactions.

Exploit Scenario

A malicious signer submits several malicious transactions and then submits an additional transaction to change the signer set of the wallet to be empty, causing the threshold to be changed to zero. The malicious signer tricks the other signers into approving the transaction that changes the signer set, allowing him to approve all of his previously submitted malicious transactions.

Recommendations

Short term, update the `set_signers` function to call the `require_valid_threshold` function whenever the threshold is changed, preventing an empty signer set with a threshold of zero.

2. Multisig constructor and associated functions allow signers to be zero	
Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EQMS-2
Target: `starknet-multisig/contracts/Multisig.cairo`	

Description

The `Multisig` contract's constructor takes as input the wallet's signer set and threshold and creates a new `Multisig` contract. Naturally, this function checks the inputs to ensure that the wallet's threshold is valid (i.e., it is at least one and is less than or equal to the number of signers) and that the signers in the signer set are unique. However, the constructor (and other associated functions, such as `set_signers`, which is used to change the signer set) do not prevent the contract from containing a signer with the value of zero.

Throughout the `Multisig` contract, the `get_caller_address` syscall is used to identify the caller of the contract. The caller's address is used for various purposes, such as to check the caller against the signer set to ensure that only a valid signer is calling a particular function. However, in StarkNet, it is currently possible for anyone to directly call a contract without using an account contract, in which case `get_caller_address` will return zero. This means that if one of the signers in the signer set is set to zero, which is currently allowed by the `Multisig` contract, then anyone can call the contract directly and be accepted as a valid signer.

Note that when users call a contract directly without an account contract, they cannot pay any fees. Fees are not currently mandatory, but there are plans to make them mandatory soon. Once that is the case, this finding will no longer be a risk.

Exploit Scenario

A multisignature wallet is set up with three signers, one of which is a zero-address signer, and with a threshold of one. A malicious user identifies this wallet and submits and executes transactions as the zero-address signer by calling the contract directly without an account contract.

Recommendations

Short term, add checks to the `Multisig` constructor and all functions that change the signer set to ensure that all signer values are nonzero.

Long term, integrate Amarna into the project's continuous integration process.

3. Multisig contract does not use the latest version of Cairo

Severity: **Informational**	Difficulty: **Undetermined**
Type: Patching	Finding ID: TOB-EQMS-3
Target: `starknet-multisig`	

Description

Cairo is a relatively new STARK-based language for writing provable programs. Therefore, Cairo is an immature language with a unique design that often results in implementation errors; however, Cairo is improving with every new version released. The latest Cairo language release at the time of writing (version 0.10.0), for instance, changes an important behavior in Cairo. Previously, importing a file into a smart contract would also import all `external` and `view` functions in that file. Now, with Cairo version 0.10.0, those functions have to be explicitly imported in the main compiled file to be usable in the contract.

As the Cairo language continues to mature and evolve, it is important to keep Cairo smart contracts up-to-date with these improvements. For the foreseeable future, Cairo releases will likely include features and improvements that help prevent misuse of the language. Additionally, the more up-to-date a Cairo smart contract is, the easier it will be to update it with each new release.

At the time of writing this issue, the codebase has not been updated to the latest version of Cairo.

Recommendations

Short term, update the `Multisig` contract to version 0.10.0, or whichever version is the latest release when this issue is addressed. The `cairo-migrate` script can be used to convert old code to the new syntax.

Long term, monitor the development of the Cairo language and ensure that all Cairo smart contracts are updated with each major release, especially releases that make the language safer to use.

4. require_tx_exists accepts negative nonces

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EQMS-4
Target: `starknet-multisig/contracts/Multisig.cairo`	

Description

The `require_tx_exists` function in the `Multisig` contract checks whether a nonce is the nonce of an existing transaction. Because nonces start at zero and increment once with each new transaction, the function compares the candidate nonce with the `next_nonce` value to check the nonce's validity.

```
# @dev Requires that the transaction exists. Reverts if the tx doesn't exist
# @param nonce: Nonce of the transaction in question
func require_tx_exists{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
range_check_ptr}(
    nonce : felt
):
    let (next_nonce) = _next_nonce.read()
    with_attr error_message("Transaction does not exist"):
        assert_lt(nonce, next_nonce)
    end
    return ()
end
```

Figure 4.1: starknet-multisig/contracts/Multisig.cairo#L138-L148

However, the function does not check that the nonce is nonnegative: the `assert_lt` function will accept negative nonces that are not associated with an existing transaction.

This is currently not a security issue because all current uses of the `require_tx_exists` function are paired with a call to the `require_tx_valid` function, which checks that the nonce is greater than or equal to a value that is never negative.

Recommendations

Short term, add a check to verify that the transaction nonce is nonnegative.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Quality Recommendations

The following recommendation is not associated with a specific vulnerability. However, it will enhance the code's readability and may prevent the introduction of vulnerabilities in the future.

- **Unused import in `Multisig.cairo`.** The `assert_not_zero` function is imported in line 5 of the `Multisig.cairo` file, but it is never used. Consider either using it for its originally intended purpose or removing it.