Security Review Report
NM-0191 Braavos Account

NETHERMIND
SECURITY

(Apr 18, 2024)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the Braavos Account. It is a smart wallet implementation designed for the Starknet ecosystem, taking advantage of the built-in account abstraction features from the Starknet network. All wallets have a single stark signer, with the option to add any number of strong signers using hardware wallets or Webauthn, which allow for more granular security features. The most notable features are a multisig option and daily withdrawal limits, where the value change over a day for specific tokens is limited. In the case of a phishing attack or a compromised signer, financial damage can be limited to the withdrawal limit. There are different daily limits depending on the signer strength, with stark lowest, followed by a strong signer, and then multisig.

Braavos Multi-Owner Account is a multisig wallet implementation. The MOA account has multiple signers, each being a StarkNet smart wallet address. Signers can propose transactions, which can then be confirmed by other signers either through separate transactions or all at once by collecting signatures off-chain. When enough approval signatures have been met to pass a specified threshold, the pending transaction will execute.

This audit has been completed following the security review of the Braavos Account contracts (NM-0125), of which many core components are shared. This means that most of a focus was placed on MOA contracts compononets as a result.

The security review started at the commit `e8753ad1eaba0f96abe1f85684c27c3223eaa062`, which is the approved commit of the changes made during the previous review (NM-0125) that were squashed. The changes introduced after that commit were the main focus of this security review. At the end of this engagement all intermediate development commits were squashed into the final, approved commit hash `696405f83c040bb16f102e3cc3274e488a410b4c`, which corresponds to the following class hashes on Starknet Mainnet:

- Braavos Base Account: `0x013bfe114fb1cf405bfc3a7f8dbe2d91db146c17521d40dcf57e16d6b59fa8e`
- Braavos Account: `0x00816dd0297efc55dc1e7559020a3a825e81ef734b558f03c83325d4da7e6253`
- Braavos MOA Account: `0x041bf1e71792aecb9df3e9d04e1540091c5e13122a731e02bec588f71dc1a5c3`

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract, and (d) creation of test cases. **Along this document, we report** 4 points of attention, where all are classified as `Informational` or `Best Practices`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 presents the commits introduced. Section 4 summarizes the issues. Section 5 presents the system overview. Section 6 discusses the risk rating methodology. Section 7 details the issues. Section 8 highlights a potential issue based on future intended features. Section 9 discusses the documentation provided by the client for this audit. Section 10 presents the compilation, tests, and automated tests. Section 11 concludes the document.

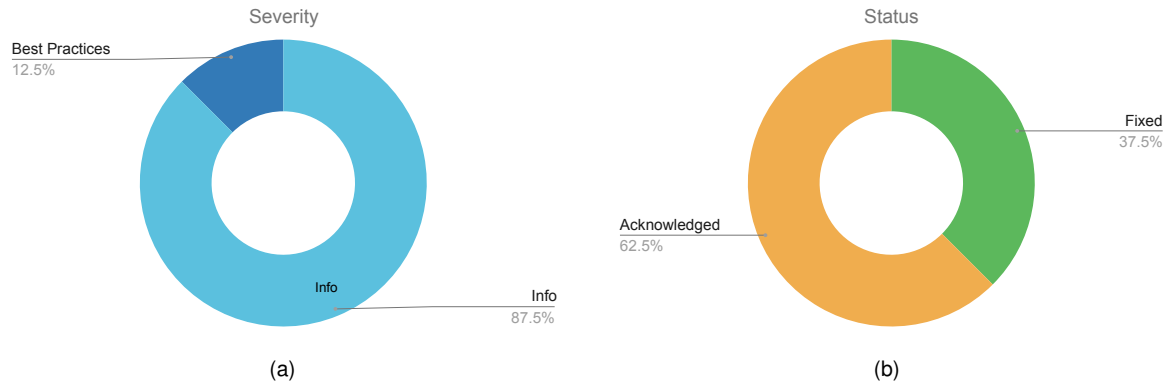

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (7), **Best Practices** (1). **Distribution of status: Fixed** (3), **Acknowledged** (5), **Mitigated** (0), **Unresolved** (0), **Partially Fixed** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Mar 5, 2024
Response from Client	Regular responses during audit engagement
Final Report	-
Repository	myBraavos/braavos-account-cairo
Commit (Audit)	e8753ad1eaba0f96abe1f85684c27c3223eaa062
Commit (Final)	696405f83c040bb16f102e3cc3274e488a410b4c
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/account.cairo	1	0	0.0%	0	1
2	src/utils.cairo	5	0	0.0%	0	5
3	src/dwl.cairo	3	0	0.0%	0	3
4	src/outside_execution.cairo	3	0	0.0%	0	3
5	src/upgradable.cairo	2	0	0.0%	0	2
6	src/transactions.cairo	4	0	0.0%	0	4
7	src/presets.cairo	3	0	0.0%	0	3
8	src/introspection.cairo	2	0	0.0%	0	2
9	src/signers.cairo	7	0	0.0%	0	7
10	src/lib.cairo	10	0	0.0%	0	10
11	src/upgradable/interface.cairo	10	2	20.0%	4	16
12	src/upgradable/upgradable.cairo	225	16	7.1%	24	265
13	src/utils/utils.cairo	205	4	2.0%	19	228
14	src/utils/snip12.cairo	35	1	2.9%	6	42
15	src/utils/hash.cairo	45	3	6.7%	5	53
16	src/utils/arrays.cairo	18	0	0.0%	1	19
17	src/utils/asserts.cairo	4	0	0.0%	1	5
18	src/introspection/interface.cairo	6	3	50.0%	2	11
19	src/introspection/src5.cairo	51	6	11.8%	7	64
20	src/transactions/moa_tx_hash.cairo	23	0	0.0%	5	28
21	src/transactions/interface.cairo	41	5	12.2%	5	51
22	src/transactions/pending_txn.cairo	219	58	26.5%	26	303
23	src/transactions/daily_txn_limit.cairo	37	23	62.2%	8	68
24	src/dwl/rate_service.cairo	385	69	17.9%	44	498
25	src/dwl/dwl.cairo	506	90	17.8%	60	656
26	src/dwl/interface.cairo	181	42	23.2%	25	248
27	src/outside_execution/outside_execution.cairo	79	5	6.3%	12	96
28	src/outside_execution/interface.cairo	17	10	58.8%	4	31
29	src/outside_execution/hash.cairo	33	0	0.0%	5	38
30	src/account/interface.cairo	46	11	23.9%	12	69
31	src/presets/braavos_base_account.cairo	76	24	31.6%	18	118
32	src/presets/braavos_account.cairo	547	132	24.1%	73	752
33	src/presets/braavos_moa_account.cairo	262	93	35.5%	39	394
34	src/signers/signer_management.cairo	352	45	12.8%	36	433
35	src/signers/signer_address_mgt.cairo	169	12	7.1%	17	198
36	src/signers/multisig.cairo	74	19	25.7%	12	105
37	src/signers/interface.cairo	79	12	15.2%	15	106
38	src/signers/signer_type.cairo	51	2	3.9%	7	60
39	src/signers/signers.cairo	329	39	11.9%	40	408
40	src/signers/moa_signer_management.cairo	143	31	21.7%	18	192
	Total	**4288**	**757**	**17.7%**	**550**	**5595**

3 Summary of Issues

	Finding	Severity	Update
1	Missing transaction signature field validations allows for DOS	Info	Fixed
2	DOS of propose and execute transactions	Info	Fixed
3	Stark pubkey or signer address can reused by multiple signers	Info	Acknowledged
4	Incorrect calldata structure comment	Best Practices	Fixed
5	Deferred removals not included in storage migration	Info	Acknowledged
6	Outside execution transactions	Info	Acknowledged
7	The `IntoOrPanic` logic for `SpanFelt252IntoU256` is non-injective	Info	Acknowledged
8	WebAuthn signature may contain multiple transaction hashes	Info	Acknowledged

4 System Overview

The **Braavos Base Account** module contains contract creation and initialization logic. It allows for both UDC and non-UDC deployment. A deployed wallet address is dependent only on the stark public key but still allows for contract initialization by passing data in signature, not as constructor arguments. Independence from initialization data protects the user in case of lost trusted hardware during deployment after the funds are sent to the determined wallet address. It also allows for setting current token rates and delegation of wallet deployment to third parties.

The **Braavos Account** is the main module in which validation and execution of the transaction are performed. Validation is split between two functions `__validate__(...)` and `__execute__(...)` to allow for different signer validation depending on transaction type and amount of funds spent.

The **Daily Withdrawal Limit** module is responsible for setting daily limits, storing and updating `ETH` and `STRK` rates, checking spending range of transaction, expressing transaction value in threshold currency (`USDC`). If the calls meet whitelist requirements, the balance of tokens is checked before and after the calls are executed. The difference is then calculated and added to the daily withdrawal amount.

The **Outside Execution** module implements SNIP9. It allows wallet owner(s) to sign a transaction that later can be executed by a trusted third-party contract through additional entrypoint `execute_from_outside_v2(...)`. These transactions come with additional parameters specifing who can execute them, and the valid time range in which it must be done.

The **Braavos Multi-Owner Account (MOA)** module contains the main logic of Multisig transactions. It stores `guid` numbers which represent owners accounts. It has one entry point through which owners may do transactions: `__validate__(...)` and `__execute__(...)`. In the validation phase, it checks the following properties: Stark signatures validity, daily limits, uniqueness of the signatures for pending transactions, transaction cost, and if the signed transaction should be executed. Then, on the execute part, MOA calls `is_valid_signature(...)` on owners' account contracts to check the validity of provided signatures (Stark, hardware, WebAuthn) and then executes the transactions. This contract is designed to protect the MOA from burning gas by a potentially single malicious owner by gas limits check on validation.

The **Transactions** module handles all transaction proposal and daily limit features. It tracks how many transactions a particular signer has initiated daily and ensures they are within their limit. Confirmations for each transaction are checked and updated here, including logic related to transaction execution if the required threshold of confirmations has been met. This module also contains helper functions related to safe gas consumption with fee checks.

The **Signers** module is responsible for the management of wallet signers, configuring multisig parameters, and signature validation. The MOA signer implementation also has parser logic to extract signer data out of the transaction signature field.

The **Upgradeable** module is responsible for wallet implementation upgrades and storage migration.

The **Utils** module with functions required to manipulate data, e.g., prepare as input to hashing function.

The **Introspection** module implements SNIP5 and SNIP6, which is similar to the Ethereum EIP-165. It allows other contracts to detect if the Braavos wallet supports particular interfaces. This is used by the upgradability feature to ensure that the upgrade will be to a compatible implementation.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Info] Missing transaction signature field validations allows for DOS

File(s): `braavos_moa_account.cairo`

Description: When a pending transaction is proposed or confirmed on the Braavos MOA Account, some number of signatures are provided and then verified to ensure the given transaction is valid, and no calldata has been tampered with. This approach is secure when all signature data is checked during `__validate__`, since said checks can catch any modified signature data.

The following is a breakdown of the structure of signature data:
`[sigtype] [address] [pubkey] [preamble_r] [preamble_s] [extsiglen] [extsig]`

From these fields, the `__validate__` function ensures that the following are valid:

- `sigtype`: Checked by `resolve_signers_from_sig`, must be equal to zero;
- `address`: Combined with `pubkey` to create `MoaExtSigner`;
- `pubkey`: Combined with `address` to create `MoaExtSigner`;
- `preamble_r`: Must be valid signature field `r` for `pubkey`'s signature of the pending transaction hash;
- `preamble_s`: Must be valid signature field `s` for `pubkey`'s signature of the pending transaction hash;
- `extsiglen`: Cannot be zero;
- `extsig`: No validation;

The `MoaExtSigner` object then has the following validations:

- Signer must exist;
- Must be valid for given preamble signature on pending transaction hash;
- Must not have already confirmed the pending transaction;
- If signer is proposer or executing single confirmation transaction, must not have exceeded the daily transaction limit;

The `extsig` field is not validated but is used in `__execute__` and passed as calldata to `is_valid_signature` on the signer wallet address. This field can be modified by anyone to be an invalid signature, but the change will only be detected in `__execute__` where it will revert. By reaching `__execute__` the nonce will have increased, gas will have been spent, and the victim proposer's limit will have increased. The increasing nonce will then cause the original transaction to revert since the calculated MOA tx hash will have changed, making the original signature invalid.

An attacker can DOS the wallet and prevent any interactions with it using this approach. There are different impacts depending on if the victim transaction is a proposal or a confirmation:

- When proposal transaction, provided the victim has not reached their daily tx limit, the attacker must continuously read the mempool and frontrun every transaction. The attacker's modified transaction will reach `__execute__` and fail, increasing the nonce. When the victim's transaction arrives, the given signature will be for the MOA tx with the previous nonce, but since the nonce has been increased by the attacker, this signature verification will fail. This can continue up until the daily transaction limit for the victim is reached;

- When confirmation transaction, once a victim submits their confirmation, the attacker can frontrun with 25 transactions at once, all with modified external signatures. This instantly causes the victim signer to reach their daily tx limit, and the signer cannot complete any actions (confirm or propose) for up to 24 hours;

This issue also applies to the `extsiglen` field; although it has validations, it is under-constrained. An attacker can modify the `extsiglen` to be shorter than expected, which will cause signature verification to fail. This field is only exploitable when there is one signature, however (non-multi-signer proposal or basic confirmation), since a modified `extsiglen` with multiple signatures would cause the following signatures to be parsed incorrectly.

> **Finding note**
>
> When the Starknet sequencers become decentralized, transactions will no longer be first-come-first-serve, and it will be possible to view transactions in the mempool before execution. Attackers will be able to observe, modify, and frontrun existing transactions. Currently, the Starknet sequencers are centralized, and this is not possible. The following explores an attack vector that is not currently exploitable but will be when sequencers are decentralized. For this reason, the finding now has a severity of `Info`, but left unfixed; it may have been considered a much higher severity in the future.

Recommendation(s): Consider signing the `extsiglen` and `extsig` fields in the transaction signature and then validating in `__validate__` to ensure the external signature fields have not been tampered with.

Status: Fixed.

Update from the client: Acknowledged and fixed.

6.2 [Info] DOS of propose and execute transactions

File(s): `braavos_moa_account.cairo`

Description: The MOA account transaction may be reverted during the propose-execute flow by any malicious actor. The MOA transaction my be proposed and immediately executed if the number of signers satisfies the threshold. However, any malicious actor watching the mempool can do following steps to invalidate incoming transaction `A`:

- extract one signature in the `signature` field of transaction `A`, e.g. if there are 6 signatures take the first one ;
- create the same transaction `A` but with only one valid signature, using it as a proposal transaction ;
- front-run the original transaction ;

In such scenario, the transaction posted by the malicious actor would be a propose transaction, and wouldn't execute the calls, but would change the following:

- nonce of a MOA would be incremented by `1` ;
- the `guid` of the owner of used signature would be marked in the mapping `_is_confirmed(...)` ;

Those changes would make the original transaction invalid, since it was signed for different nonce, and one of the signers is already marked as used. While this attack can be performed by any actor watching the mempool, the front-running can't be performed by offering the Sequencer more gas, since the malicious actor must use the same gas limits in calldata as in original transaction. This attack vector however may be possible in a future, when block proposing and transaction ordering is decentralized between the network of Sequencers. Then the malicious actor could watch a mempool and front-run the original transaction by publishing the transaction to a private mempool, which could execute the transaction faster. The attack vector theoretically allows for DOS of propose-execute transaction flow, which may be dangerous in time-sensitive situations. However, due to limitations described above, we rank the severity as low.

Finding note

When the Starknet sequencers become decentralized, transactions will no longer be first-come-first-serve, and it will be possible to view transactions in the mempool before execution. Attackers will be able to observe, modify, and frontrun existing transactions. Currently, the Starknet sequencers are centralized, and this is not possible. The following explores an attack vector that is not currently exploitable but will be when sequencers are decentralized. For this reason, the finding now has a severity of `Info`, but left unfixed; it may have been considered a higher severity in the future.

Recommendation(s): This attack vector could be mitigated by adding to the pending transaction hash a number of of signers proposing a transactions, so that a malicious actor can't block the execution of the desired calls. We present an example of proposed change in `__validate__(...)` function below, with skipped part marked as `...`:

```
fn __validate__(ref self: ContractState, calls: Span<Call>) -> felt252 {
        ...
    if (_is_verifier_tx(calls)) {
-       pending_tx_hash = calculate_moa_tx_hash(proposer_guid, pending_nonce, inner_calls);
+      pending_tx_hash = calculate_moa_tx_hash(proposer_guid, pending_nonce, sig_number, inner_calls);
        ...
    } else {
        // Tx proposer flow
        PendingTransactions::_assert_max_fee_is_set(calls);
        pending_tx_hash =
-            calculate_moa_tx_hash(signers.at(0).signer.guid(), tx_info.nonce, calls);
+           calculate_moa_tx_hash(signers.at(0).signer.guid(), tx_info.nonce, signers.len(), calls);
    }
   ...
}
```

Status: Fixed.

Update from the client: Acknowledged and fixed.

6.3 [Info] Stark pubkey or signer address can reused by multiple signers

File(s): `signers.cairo`

Description: The guid of an `MoaSigner` is calculated by hashing the Stark pubkey and the signer wallet address, as shown below:

```
#[derive(Copy, Drop, Serde, starknet::Store)]
struct MoaSigner {
    address: ContractAddress,
    pub_key: felt252,
}

fn guid(self: @MoaSigner) -> felt252 {
    let mut serialized = ArrayTrait::new();
    self.serialize(ref serialized);
    poseidon::poseidon_hash_span(serialized.span())
}
```

A newly added signer can use the pubkey of an existing signer, as long as the `address` field is unique, since the hash output will result in a different guid. The same can apply to the `address` field, where two signers can share the same address as long as their pubkeys are different.

The function `_add_signers` partially prevents this, by checking that each signer address being added must have a unique contract address, but these comparisons only apply to the new signers, and does not consider existing signers. Therefore a signer could be added with a reused `address` field as long as the address being copied has been added before the current call.

For a signer with a reused address or pubkey to be added to the MOA account, they must be approved by all existing signers, making this an unlikey scenario. The impacts of having a shared signer or pubkey are negligible, since all validations will be tied to the original signer or wallet address.

Recommendation(s): As this behavior is unlikely to occur, and the impacts aren't significant, no actions by the Braavos team are needed. However, allowing shared contracts or pubkeys between signers is still an unusual feature, so this is presented as an informational finding.

Status: Acknowledged.

Update from the client: Acknowledged.

6.4 [Best Practices] Incorrect calldata structure comment

File(s): `braavos_moa_account.cairo`

Description: The function `_extract_fee_limits` contains a series of comments explaining the calldata structure of `sign_pending_multisig_transaction`. The comments state that `pending_nonce` is the first argument, followed by the `proposer_guid`, as shown below:

```
// calldata structure of sign pending is as follows:
// 0: length of calldata span, 1: pending nonce, 2: proposer guid
// 3: first call to, 4: first call selector, 5: first call calldata length
// 6: index of the calldata of assert_max_fee which is of length 4
```

The actual structure of the calldata for `sign_pending_multisig_transaction` is as follows, where the `proposer_guid` is the first argument, and the `pending_nonce` is the second arg:

```
fn sign_pending_multisig_transaction(
    ref self: ComponentState<TContractState>,
    proposer_guid: felt252,
    pending_nonce: felt252,
    calls: Span<Call>
) -> Array<Span<felt252>> {...}
```

Recommendation(s): Consider correcting the ordering of arguments in the highlighted comment.

Status: Fixed.

Update from the client: Acknowledged and fixed.

7 Issues from the previous security review

This section presents acknowledged issues from the previous security review (NM-0125) that are still in the codebase.

7.1 [Info] Deferred removals not included in storage migration

File(s): `upgradable.cairo`

Description: When migrating storage from a Braavos wallet on version `000.000.011` to the current version `001.000.000`, the following are changed in `migrate_storage` to meet the new storage structure requirements:

- Stark signer;
- Secp256r1 signer (if exists);
- Multisig threshold (if the previous implementation uses multisig);

Deferred removals on the previous version are not migrated. Under normal circumstances, a user would not have a deferred removal during migration, as the only reason for a deferred removal is in the case of a lost hardware wallet, but a lost hardware wallet would mean that the multisig requirement cannot be met, making a migration impossible.

There is an unlikely case where a user has both a stark signer and hardware signer on their old wallet version, but the multisig had been disabled via `disable_multisig` prior to the hardware wallet being lost. This is a possible scenario where a user may attempt to undergo a storage migration during a deferred removal.

Since deferred removals are not migrated to the new wallet version, any existing deferred removal will be lost and cannot be completed.

Recommendation(s): This doesn't pose a security risk, however it should be made clear to users that a deferred removal of a hardware wallet will not work if the process is started on version `000.000.011` and completed post-migration on version `001.000.000`.

Status: Acknowledged

Update from the client: Acknowledged. Deferred removal are for a case a user lost access to his hardware signer. If a user had successfully executed an upgrade it means he had access to his HWS device making the deferred removal request redundant as he could just choose to remove HWS without deferral. Since we don't see a reason to support this use-case we decided not to migrate the deferred request and save the storage write gas.

7.2 [Info] Outside execution transactions

File(s): outside_execution.cairo

Description: The outside execution (OE) module allows the wallet owners to pre-sign the transaction and share it with a trusted party, which is able to execute this transaction in the future on behalf of the wallet by calling `execute_from_outside_v2(...)`. There are, however, some points of concern with the current design. Below, we list those concerns:

- OE transactions are not bound to the current wallet version. Since the wallet is an upgradable contract, its implementation may change. This creates a possibility of signing a transaction for one wallet version but executing it (by a third party) on the new wallet version ;
- The OE mechanism doesn't allow the invalidation of all previously signed transactions. There is also no mechanism to pause `execute_from_outside_v2(...)` entry point. But wallet owners can still invalidate pre-signed OE transactions by burning the nonce, i.e., signing and executing OE transactions with the nonce of other transaction ;

Those areas of concern may result in unwanted behavior under certain conditions but do not pose a direct threat to the funds in the wallet.

Recommendation(s): Since the OE implementation follows the standard described in SNIP-9: Outside execution, we don't recommend to introduce changes to the implementation, which would result in deviation from the standard. However, we strongly recommend informing users about the possible issues/misuses in the documentation.

Status: Acknowledged

Update from the client: Acknowledged, we agree that these issues do not pose a risk at the moment. We will take note of outside execution transaction backward compatibility issues when working on new account versions.

7.3 [Info] The `IntoOrPanic` logic for `SpanFelt252IntoU256` is non-injective

File(s): `utils.cairo`

Description: The `utils` module defines `IntoOrPanic` logic for `SpanFelt252IntoU256`, which is used to convert a `span<felt252>` into a `u256`, and will panic otherwise. The input is expected to be a span with 8 elements, each containing 32 bits of data. These individual elements are then combined together to give the resulting `u256`, as shown below:

```
impl SpanFelt252IntoU256 of IntoOrPanic<Span<felt252>, u256> {
    fn into_or_panic(self: Span<felt252>) -> u256 {
        assert(self.len() == 8, 'INVALID_FELT252s_U256_CONV_LEN');
        u256 {
            high: ((*self.at(0)).try_into().unwrap() * 0x100000000000000000000000_u128
                + (*self.at(1)).try_into().unwrap() * 0x1000000000000000_u128
                + (*self.at(2)).try_into().unwrap() * 0x100000000_u128
                + (*self.at(3)).try_into().unwrap()),
            low: ((*self.at(4)).try_into().unwrap() * 0x100000000000000000000000_u128
                + (*self.at(5)).try_into().unwrap() * 0x1000000000000000_u128
                + (*self.at(6)).try_into().unwrap() * 0x100000000_u128
                + (*self.at(7)).try_into().unwrap())
        }
    }
}
```

This logic assumes that each `felt252` element in the span will bit in 32 bits, however this is not checked and cannot be guaranteed. A `felt252` with 64 bits can be a part of the span, and the extra bits will affect the 32 more-significant bits. An example is shown below:

```
#Normal case
input1 = 0x11111111
input2 = 0x22222222

         0x0000000011111111
       + 0x2222222200000000
       = 0x2222222211111111

# Edge case
input1 = 0x2222222211111111
input2 = 0x00000000

         0x2222222211111111
       + 0x0000000000000000
       = 0x2222222211111111
```

This allows for multiple unique `span<felt252>` inputs to result in the same output. After assessing the codebase, this edge case poses no risk to wallet logic as the `into_or_panic` is only used on return data from `SHA256_CAIRO_0_LIB`, which is expected to only return a maximum of 32 bits per span element.

Although it currently poses no risk, it should be noted that if this function were to be used in the future to process user input, multiple inputs could lead to the same output. If used for hashing or cryptographic signatures, this may lead to security issues.

Recommendation(s): As this function is not used in a way that could lead to a security risk currently, no action needs to be taken.

Status: Acknowledged

Update from the client: Acknowledged, we agree that this issue does not pose a risk at the moment but will be taken into account in any future use of this logic.

7.4 [Info] WebAuthn signature may contain multiple transaction hashes

File(s): `signers.cairo`

Description: The WebAuthn signature may contain multiple challenges (tx hashes). Therefore, the valid signer can sign future transactions by computing the transaction hashes and extending the challenge so that it includes multiple transaction hashes. Below, we present an example of client data modification:

```
{
  "type":"webauthn.get",
- "challenge":tx_hash,
+ "challenge":tx_hash || tx_hash_1 || tx_hash_2,
  "origin":"http://beefdead",
  "crossOrigin":false}
```

Instead of one transaction hash, the client data contains three transaction hashes. The unrestricted fields in the signature: `challenge_offset` and `challenge_len`, may be modified to reuse this signature by pointing to a different part of "challenge" as a transaction hash. While it does not pose a direct security issue, it may break the assumptions that:

- WebAuthn signer signs each transaction separately ;
- WebAtuhn signer is present at the time of each transaction signature ;

This issue may or may not lead to a vulnerability or unwanted behavior depending on off-chain wallet implementation parts, which are not part of scope.

Recommendation(s): Consider ensuring that the `challenge_offset` and `challenge_len` fields are in a valid range of values.

Status: Acknowledged

Update from the client: Acknowledged, we believe that his does not pose a risk. Additionally there are several limitations that mitigate this issue, such as asserting that the challenge section is surrounded by double quotes.

8 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about the Braavos Account Wallet documentation

The documentation for the Braavos Account Wallet is contained in the README of the project's Github. It is written for developers, presenting each core component of the wallet, including deployment, signers, multisig, deferred removals, daily withdrawal limits, and outside execution.
The information is presented in a very concise and technical manner, with well-written explanations and references where necessary. It also features a section explaining development dependencies and instructions on how to build and test the code.
Code comments are also of high quality, with explanations for every function describing the purpose, context, and situations where the function will be called. Other than functions, some comments exist in other areas of the code where extra information is necessary, allowing readers to understand the codebase at a faster pace.

9 Test Suite Evaluation

9.1 Contracts Compilation

```
user@hardware:~/braavos$ scarb --version
scarb 2.5.1 (9d216f5b0 2024-01-29)
cairo: 2.5.1 (https://crates.io/crates/cairo-lang-compiler/2.5.1)
sierra: 1.4.0

user@hardware:~/braavos$ scarb build
   Compiling braavos_account v1.0.0 (/home/user/braavos/Scarb.toml)
    Finished release target(s) in 7 seconds
```

9.2 Tests Output

```
user@hardware:~/braavos$ pytest
============================================ test session starts ============================================
platform linux -- Python 3.9.13, pytest-7.2.0, pluggy-1.0.0 -- /home/user/.pyenv/versions/3.9.13/bin/python3.9
cachedir: .pytest_cache
rootdir: /home/user/braavos, configfile: pytest.ini
plugins: anyio-3.7.0, typeguard-2.13.3, web3-6.14.0, asyncio-0.20.1, profiling-1.7.0, xdist-3.1.0
asyncio: mode=strict
collected 154 items

test_Account.py::test_deployment[basic_stark_signer] PASSED
test_Account.py::test_deployment[with_secp256r1_no_multisig] PASSED
test_Account.py::test_deployment[with_secp256r1_multisig] PASSED
test_Account.py::test_deployment[with_webauthn_no_multisig] PASSED
test_Account.py::test_deployment[with_webauthn_multisig] PASSED
test_Account.py::test_deployment[with_secp256r1_no_multisig_with_thresh] PASSED
test_Account.py::test_deployment[with_secp256r1_multisig_with_thresh] PASSED
test_Account.py::test_deployment[with_webauthn_secp256r1_no_multisig_with_thresh] PASSED
test_Account.py::test_deployment[with_webauthn_secp256r1_multisig_with_thresh] PASSED
test_Account.py::test_deployment_from_UDC PASSED
test_Account.py::test_upgrade[src6_supported] PASSED
test_Account.py::test_upgrade[src6_not_supported] PASSED
test_Account.py::test_invalid_account_deployment[stark_pub_key_is_0] PASSED
test_Account.py::test_invalid_account_deployment[basic_stark_wrong_sig] PASSED
test_Account.py::test_invalid_account_deployment[basic_stark_rs_zero_sig] PASSED
test_Account.py::test_invalid_account_deployment[multisig_1_with_only_stark] PASSED
test_Account.py::test_invalid_account_deployment[multisig_2_with_only_stark] PASSED
test_Account.py::test_invalid_account_deployment[multisig_1_with_secp256r1] PASSED
test_Account.py::test_invalid_account_deployment[multisig_3_with_secp256r1] PASSED
test_Account.py::test_invalid_account_deployment[multisig_1_with_secp256r1_webauthn] PASSED
test_Account.py::test_invalid_account_deployment[multisig_3_with_secp256r1_webauthn] PASSED
test_Account.py::test_invalid_account_deployment[invalid_secp256r1_pubk] PASSED
test_Account.py::test_invalid_account_deployment[invalid_secp256r1_pubk_x] PASSED
test_Account.py::test_invalid_account_deployment[invalid_secp256r1_pubk_y] PASSED
test_Account.py::test_invalid_account_deployment[invalid_secp256r1_pubk_webauthn] PASSED
test_Account.py::test_invalid_account_deployment[invalid_secp256r1_pubk_x_webauthn] PASSED
test_Account.py::test_invalid_account_deployment[invalid_secp256r1_pubk_y_webauthn] PASSED
test_Account.py::test_invalid_account_deployment[low_withdrawal_limit_set_with_only_stark] PASSED
test_Account.py::test_invalid_account_deployment[low_withdrawal_limit_set_without_fee_rate] PASSED
test_Account.py::test_fail_initializer_after_deployment PASSED
test_Account.py::test_add_secp256r1_signer[with_secp256r1_no_multisig] PASSED
test_Account.py::test_add_secp256r1_signer[with_secp256r1_multisig] PASSED
test_Account.py::test_add_secp256r1_signer[with_secp256r1_and_webauthn_no_multisig] PASSED
test_Account.py::test_add_secp256r1_signer[with_secp256r1_and_webauthn_multisig] PASSED
test_Account.py::test_add_webauthn_signer[with_webauthn_no_multisig] PASSED
test_Account.py::test_add_webauthn_signer[with_webauthn_and_hws_no_multisig] PASSED
test_Account.py::test_add_webauthn_signer[with_webauthn_multisig] PASSED
test_Account.py::test_add_webauthn_signer[with_webauthn_and_hws_multisig] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_no_multisig] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_multisig] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_no_multisig_with_webauthn] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_multisig_with_webauthn] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_no_multisig_with_thresh] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_multisig_with_thresh] PASSED
```

```
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_no_multisig_with_webauthn_with_thresh] PASSED
test_Account.py::test_remove_secp256r1_signer[with_secp256r1_multisig_with_webauthn_with_thresh] PASSED
test_Account.py::test_remove_webauthn_signer[with_webauthn_no_multisig] PASSED
test_Account.py::test_remove_webauthn_signer[with_webauthn_multisig] PASSED
test_Account.py::test_remove_webauthn_signer[with_webauthn_no_multisig_with_hws] PASSED
test_Account.py::test_remove_webauthn_signer[with_webauthn_multisig_with_hws] PASSED
test_Account.py::test_change_secp256r1_signer[with_secp256r1_no_multisig] PASSED
test_Account.py::test_change_secp256r1_signer[with_secp256r1_multisig] PASSED
test_Account.py::test_change_weauthn_signer[with_webauthn_no_multisig] PASSED
test_Account.py::test_change_weauthn_signer[with_webauthn_multisig] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_secp256r1_no_multisig_custom_time_delay] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_secp256r1_multisig] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_webauthn_secp256r1_no_multisig] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_webauthn_secp256r1_multisig] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_secp256r1_no_multisig_with_thresh] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_secp256r1_multisig_with_thresh] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_webauthn_secp256r1_no_multisig_with_thresh] PASSED
test_Account.py::test_deferred_remove_secp256r1_signer[with_webauthn_secp256r1_multisig_with_thresh] PASSED
test_Account.py::test_deferred_remove_all_signers[no_multisig] PASSED
test_Account.py::test_deferred_remove_all_signers[multisig] PASSED
test_Account.py::test_cancel_deferred_remove_secp256r1_signer[with_secp256r1_no_multisig] PASSED
test_Account.py::test_cancel_deferred_remove_secp256r1_signer[with_secp256r1_multisig] PASSED
test_Account.py::test_cancel_deferred_remove_secp256r1_signer[with_webauthn_secp256r1_no_multisig] PASSED
test_Account.py::test_cancel_deferred_remove_secp256r1_signer[with_webauthn_secp256r1_multisig] PASSED
test_Account.py::test_set_execution_time_delay PASSED
test_Account.py::test_multiple_secp256r1_signers[with_secp256r1_no_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers[with_secp256r1_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers[with_webauthn_secp256r1_no_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers[with_webauthn_secp256r1_multisig] PASSED
test_Account.py::test_regenesis_upgrade[basic_stark] PASSED
test_Account.py::test_regenesis_upgrade[with_secp256r1_no_multisig] PASSED
test_Account.py::test_regenesis_upgrade[with_secp256r1_no_multisig_move_idx] PASSED
test_Account.py::test_regenesis_upgrade[with_secp256r1_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers_with_signer_change[with_secp256r1_no_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers_with_signer_change[with_secp256r1_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers_with_signer_change[with_webauthn_secp256r1_no_multisig] PASSED
test_Account.py::test_multiple_secp256r1_signers_with_signer_change[with_webauthn_secp256r1_multisig] PASSED
test_Account.py::test_multiple_hws_and_webauthn_signers_with_signer_change[multisig] PASSED
test_Account.py::test_multiple_hws_and_webauthn_signers_with_signer_change[no_multisig] PASSED
test_Account.py::test_duplicate_signers_cause_error[webauthn] PASSED
test_Account.py::test_duplicate_signers_cause_error[not_webauthn] PASSED
test_Account.py::test_setting_low_withdrawal_threshold_success[with_secp256r1_no_multisig] PASSED
test_Account.py::test_setting_low_withdrawal_threshold_success[with_secp256r1_multisig] PASSED
test_Account.py::test_setting_low_withdrawal_threshold_success[with_webauthn_secp256r1_no_multisig] PASSED
test_Account.py::test_setting_low_withdrawal_threshold_success[with_webauthn_secp256r1_multisig] PASSED
test_Account.py::test_est_fee_sig_bypass[stark_signer] PASSED
test_Account.py::test_est_fee_sig_bypass[secp256r1_no_multisig] PASSED
test_Account.py::test_est_fee_sig_bypass[webauthn_no_multisig] PASSED
test_Account.py::test_est_fee_sig_bypass[stark_secp256r1_multisig] PASSED
test_Account.py::test_est_fee_sig_bypass[stark_webauthn_multisig] PASSED
test_Account.py::test_est_fee_sig_bypass[secp256r1_webauthn_multisig] PASSED
test_Account.py::test_setting_low_withdrawal_threshold_failure[basic_stark_should_fail] PASSED
test_Account.py::test_setting_high_withdrawal_threshold_success[with_secp256r1_multisig] PASSED
test_Account.py::test_setting_high_withdrawal_threshold_failure[basic_stark_should_fail] PASSED
test_Account.py::test_setting_high_withdrawal_threshold_failure[with_secp256r1_no_multisig_should_fail] PASSED
test_Account.py::test_removing_low_threshold[with_secp256r1_no_multisig_no_high] PASSED
test_Account.py::test_removing_low_threshold[with_secp256r1_multisig_no_high] PASSED
test_Account.py::test_removing_low_threshold[with_secp256r1_multisig_with_high] PASSED
test_Account.py::test_removing_high_threshold[without_existing_lower_threshold] PASSED
test_Account.py::test_removing_high_threshold[with_existing_lower_threshold] PASSED
test_Account.py::test_pricing_contract PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_eth_with_stark_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_usdc_with_stark_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_usdc_with_stark_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_usdc_with_hws_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_high_thresh_multisig_usdc_with_hws_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_eth_with_hws_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_high_thresh_multisig_eth_with_hws_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_eth_with_hws_webauthn] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_usdc_with_stark] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_usdc_with_hws] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_usdc_with_stark] PASSED
```

```
test_Account.py::test_successful_single_range_transfer[transfer_high_thresh_multisig_usdc_with_hws] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_eth_with_stark] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_multisig_eth_with_hws] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_high_thresh_multisig_eth_with_hws] PASSED
test_Account.py::test_successful_single_range_transfer[transfer_low_thresh_eth_with_hws] PASSED
test_Account.py::test_successful_dual_threshold_transfer[transfer_usdc] PASSED
test_Account.py::test_successful_dual_threshold_transfer[transfer_eth] PASSED
test_Account.py::test_non_whitelisted_token_cannot_bypass[transfer] PASSED
test_Account.py::test_bad_calls_structure_cant_bypass PASSED
test_Account.py::test_bad_transfer_call PASSED
test_Account.py::test_changing_rate_works[transfer_lower_rate] PASSED
test_Account.py::test_changing_rate_works[transfer_higher_rate] PASSED
test_Account.py::test_changing_rate_works[transfer_lower_close_rate] PASSED
test_Account.py::test_changing_rate_works[transfer_higher_close_rate] PASSED
test_Account.py::test_transfer_amount_too_large_for_u128 PASSED
test_Account.py::test_successful_multicall[low_thresh_multisig_with_stark] PASSED
test_Account.py::test_successful_multicall[low_thresh_multisig_with_hws] PASSED
test_Account.py::test_successful_multicall[low_thresh_no_multsig_with_stark] PASSED
test_Account.py::test_successful_multicall[high_thresh_multisig_with_hws] PASSED
test_Account.py::test_successful_multicall[low_high_thresh_multisig_with_hws] PASSED
test_Account.py::test_multicall_bypass_failures PASSED
test_Account.py::test_get_required_signer PASSED
test_Account.py::test_webauthn_chromium_examples[test_case_1] PASSED
test_Account.py::test_webauthn_chromium_examples[test_case_2_additional_fields_in_cdata] PASSED
test_Account.py::test_external_entrypoints_assert_self PASSED
test_Account.py::test_outside_execution[no_second_signer_no_multisig] PASSED
test_Account.py::test_outside_execution[with_secp256r1_no_multisig] PASSED
test_Account.py::test_outside_execution[with_secp256r1_multisig] PASSED
test_Account.py::test_outside_execution[with_webauthn_no_multisig] PASSED
test_Account.py::test_outside_execution[with_webauthn_multisig] PASSED
test_Account.py::test_outside_execution_interface PASSED
test_Account.py::test_outside_execution_with_invalid_params[execute_time_in_future] PASSED
test_Account.py::test_outside_execution_with_invalid_params[execute_time_in_past] PASSED
test_Account.py::test_outside_execution_with_invalid_params[different_caller] PASSED
test_Account.py::test_outside_execution_nonce_reuse PASSED
test_Account.py::test_outside_execution_empty_sig PASSED
test_Account.py::test_outside_execution_invalid_sig PASSED
test_Account.py::test_outside_execution_self_call PASSED

==================================== 154 passed in 2483.24s (0:41:23) =====================================
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.9.13, pytest-7.2.0, pluggy-1.0.0 -- /home/dev/.pyenv/versions/3.9.13/bin/python3.9
cachedir: .pytest_cache
rootdir: /braavos/braavos-account-cairo-private, configfile: pytest.ini
plugins: anyio-3.7.0, typeguard-2.13.3, web3-6.14.0, asyncio-0.20.1, profiling-1.7.0, xdist-3.1.0
asyncio: mode=strict

e2e/moa/test_moa.py::test_deployment[signers_0_threshold_0] PASSED
e2e/moa/test_moa.py::test_deployment[signers_0_threshold_1] PASSED
e2e/moa/test_moa.py::test_deployment[signers_0_threshold_2] PASSED
e2e/moa/test_moa.py::test_deployment[signers_1_threshold_0] PASSED
e2e/moa/test_moa.py::test_deployment[signers_1_threshold_1] PASSED
e2e/moa/test_moa.py::test_deployment[signers_1_threshold_2] PASSED
e2e/moa/test_moa.py::test_deployment[signers_2_same_pubkey_threshold_0] PASSED
e2e/moa/test_moa.py::test_deployment[signers_2_same_pubkey_threshold_1] PASSED
e2e/moa/test_moa.py::test_deployment[signers_2_same_pubkey_threshold_2] PASSED
e2e/moa/test_moa.py::test_deployment[signers_2_threshold_0] PASSED
e2e/moa/test_moa.py::test_deployment[signers_2_threshold_1] PASSED
e2e/moa/test_moa.py::test_deployment[signers_2_threshold_2] PASSED
e2e/moa/test_moa.py::test_deployment[signers_3_threshold_2] PASSED
e2e/moa/test_moa.py::test_deployment_with_duplicate_accounts PASSED
e2e/moa/test_moa.py::test_invoke_without_balance[v1] PASSED
e2e/moa/test_moa.py::test_invoke_without_balance[v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_1_threshold_0_sign_0] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_0_sign_0] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_0_sign_1] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_0] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_1] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_1_0] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_0_1] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_2_sign_0] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_2_sign_0_1] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_2_sign_0_1_2] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_3_sign_0] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_3_sign_0_1_2] PASSED
e2e/moa/test_moa.py::test_invoke[signers_1_threshold_0_sign_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_0_sign_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_0_sign_1_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_1_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_1_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_2_threshold_2_sign_0_1_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_2_sign_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_2_sign_0_1_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_2_sign_0_1_2_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_3_sign_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke[signers_3_threshold_3_sign_0_1_2_v3] PASSED
e2e/moa/test_moa.py::test_invoke_with_invalid_params[signers_2_threshold_2_sign_0_0] PASSED
e2e/moa/test_moa.py::test_invoke_with_invalid_params[signers_3_threshold_3_sign_0_1] PASSED
e2e/moa/test_moa.py::test_invoke_with_invalid_params[signers_3_threshold_3_sign_0_1_1] PASSED
e2e/moa/test_moa.py::test_invoke_with_invalid_params[signers_2_threshold_2_sign_0_0_v3] PASSED
e2e/moa/test_moa.py::test_invoke_with_invalid_params[signers_3_threshold_3_sign_0_1_v3] PASSED
e2e/moa/test_moa.py::test_invoke_with_invalid_params[signers_3_threshold_3_sign_0_1_1_v3] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_missing_max_fee[v1] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_missing_max_fee[v3] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_signing_max_fee_too_high[v1] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_signing_max_fee_too_high[v3] PASSED
e2e/moa/test_moa.py::test_try_replace_max_fee[v1] PASSED
e2e/moa/test_moa.py::test_try_replace_max_fee[v3] PASSED
e2e/moa/test_moa.py::test_try_exceed_executing_max_fee[v1] PASSED
e2e/moa/test_moa.py::test_try_exceed_executing_max_fee[v3] PASSED
e2e/moa/test_moa.py::test_try_exceed_executing_max_fee_on_first_call[v1] PASSED
e2e/moa/test_moa.py::test_try_exceed_executing_max_fee_on_first_call[v3] PASSED
e2e/moa/test_moa.py::test_try_exceed_signing_max_fee_on_first_tx[v1] PASSED
e2e/moa/test_moa.py::test_try_exceed_signing_max_fee_on_first_tx[v3] PASSED
e2e/moa/test_moa.py::test_try_exceed_signing_max_fee[v1] PASSED
e2e/moa/test_moa.py::test_try_exceed_signing_max_fee[v3] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_invalid_signer[signers_1_threshold_0_sign_0] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_invalid_signer[signers_2_threshold_0_sign_0] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_invalid_signer[signers_2_threshold_0_sign_0_1] PASSED
```

```
e2e/moa/test_moa.py::test_try_invoke_with_invalid_signer[signers_2_threshold_0_sign_1_0] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_invalid_signer[signers_2_threshold_2_sign_1_0] PASSED
e2e/moa/test_moa.py::test_try_invoke_with_invalid_signer[signers_3_threshold_0_sign_1_2_0] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn[signers_2_threshold_2_sign_0_sign_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn[signers_3_threshold_3_sign_0_sign_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn[signers_3_threshold_2_sign_0_sign_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn_with_invalid_params[signers_2_ths_2_sign_0_sign_0] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn_with_invalid_params[signers_3_ths_3_sign_0_1_sign_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn_with_invalid_params[signers_3_ths_3_sign_0_sign_0_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn_with_invalid_params[signers_3_ths_3_sign_0_sign_0_1_2] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn_with_invalid_params[signers_3_ths_3_sign_0_1_sign_2] PASSED
e2e/moa/test_moa.py::test_sign_pending_multisig_txn_with_invalid_params[signers_3_ths_3_sign_0_sign_1_2] PASSED
e2e/moa/test_moa.py::test_execute_pending_in_third_tx PASSED
e2e/moa/test_moa.py::test_try_sign_pending_with_same_signer_twice PASSED
e2e/moa/test_moa.py::test_sign_executed_tx PASSED
e2e/moa/test_moa.py::test_sign_invalid_tx[invalid_fee_in_calls] PASSED
e2e/moa/test_moa.py::test_sign_invalid_tx[invalid_nonce] PASSED
e2e/moa/test_moa.py::test_sign_invalid_tx[invalid_all] PASSED
e2e/moa/test_moa.py::test_sign_without_pending PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_2_threshold_2_sign_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_2_threshold_2_sign_0_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_2_threshold_2_sign_2] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_3_threshold_3_sign_2] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_3_threshold_3_sign_1] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_3_threshold_3_sign_1_2] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_3_threshold_3_sign_1_2_reverse] PASSED
e2e/moa/test_moa.py::test_sign_pending_with_invalid_signer[signers_3_threshold_2_sign_1_2] PASSED
e2e/moa/test_moa.py::test_sign_with_duplicate_signers_in_one_tx PASSED
e2e/moa/test_moa.py::test_sign_with_duplicate_signers_in_two_tx PASSED
e2e/moa/test_moa.py::test_estimate_fee_allow_different_pending_hash PASSED
e2e/moa/test_moa.py::test_estimate_fee_allow_invalid_sig PASSED
e2e/moa/test_moa_dtl.py::test_daily_transaction_limit PASSED
e2e/moa/test_moa_dtl.py::test_dtl_with_execute_txn[signers_3_threshold_2_sign_0_sign_2] PASSED
e2e/moa/test_moa_dtl.py::test_dtl_with_execute_txn[signers_3_threshold_2_sign_1_sign_2] PASSED
e2e/moa/test_moa_dtl.py::test_dtl_with_validate_txn PASSED
e2e/moa/test_moa_dtl.py::test_dtl_reset_on_next_day PASSED
e2e/moa/test_moa_dtl.py::test_dtl_with_reverted_tx PASSED
e2e/moa/test_moa_dtl.py::test_dtl_with_not_validated_tx PASSED
e2e/moa/test_moa_dtl.py::test_get_tx_count PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_dapp_sanity PASSED
e2e/moa/test_moa_entrypoints.py::test_external_entrypoint_guards PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_non_existing_selector PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_malformed_calldata PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[as, sm] PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[dm, rs] PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[dmwe, rswe] PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[cdrs, rr, cddmr] PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[dm, cdrsr] PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[cdrsr, sm] PASSED
e2e/moa/test_moa_entrypoints.py::test_multicall_allowed_call_to_self_combinations[add_signer, getPublicKey] PASSED
e2e/moa/test_moa_entrypoints.py::test_is_valid_sig_sanity_stark_indexed PASSED
e2e/moa/test_moa_entrypoints.py::test_is_valid_sig_wrong_inner_sig PASSED
e2e/moa/test_moa_entrypoints.py::test_is_valid_sig_wrong_external_sig PASSED
e2e/moa/test_moa_entrypoints.py::test_is_valid_sig_wrong_hash_stark_indexed PASSED
e2e/moa/test_moa_entrypoints.py::test_is_valid_sig_2_sig_correct PASSED
e2e/moa/test_moa_entrypoints.py::test_is_valid_sig_2_sig_not_enough PASSED
e2e/moa/test_moa_entrypoints.py::test_declare_disabled PASSED
e2e/moa/test_moa_entrypoints.py::test_multisig_override_pending_txn PASSED
e2e/moa/test_moa_entrypoints.py::test_get_signers[signers_1_threshold_0] PASSED
e2e/moa/test_moa_entrypoints.py::test_get_signers[signers_2_threshold_0] PASSED
e2e/moa/test_moa_entrypoints.py::test_get_signers[signers_2_threshold_2] PASSED
e2e/moa/test_moa_entrypoints.py::test_get_signers[signers_2_threshold_2_reversed] PASSED
e2e/moa/test_moa_entrypoints.py::test_get_signers[signers_3_threshold_2] PASSED
e2e/moa/test_moa_entrypoints.py::test_get_signers[signers_3_threshold_3] PASSED
e2e/moa/test_moa_entrypoints.py::test_add_external_signers PASSED
e2e/moa/test_moa_entrypoints.py::test_add_same_external_signers_in_one_tx PASSED
e2e/moa/test_moa_entrypoints.py::test_add_same_external_signer_address_in_one_tx PASSED
e2e/moa/test_moa_entrypoints.py::test_add_same_external_signers_in_diff_tx PASSED
e2e/moa/test_moa_entrypoints.py::test_try_add_empty_external_signers PASSED
e2e/moa/test_moa_entrypoints.py::test_empty_sig PASSED
```

```
e2e/moa/test_moa_entrypoints.py::test_try_remove_all_signers PASSED
e2e/moa/test_moa_entrypoints.py::test_remove_external_signers[signer_0_threshold_0] PASSED
e2e/moa/test_moa_entrypoints.py::test_remove_external_signers[signer_1_threshold_0] PASSED
e2e/moa/test_moa_entrypoints.py::test_invalid_threshold_after_remove_external_signers PASSED
e2e/moa/test_moa_entrypoints.py::test_removing_duplicate_signer_guids PASSED
e2e/moa/test_moa_entrypoints.py::test_removing_nonexistent_signer_guids PASSED
e2e/moa/test_moa_entrypoints.py::test_remove_and_add_external_signers PASSED
e2e/moa/test_moa_entrypoints.py::test_try_remove_deleted_signers PASSED
e2e/moa/test_moa_entrypoints.py::test_account_with_50_signers_sign_in_1_tx PASSED
e2e/moa/test_moa_entrypoints.py::test_account_with_50_signers_sign_in_2_tx_partial PASSED
e2e/moa/test_moa_entrypoints.py::test_upgrade[src6_supported] PASSED
e2e/moa/test_moa_entrypoints.py::test_upgrade[src6_not_supported] PASSED
e2e/moa/test_moa_entrypoints.py::test_regenesis_upgrade[4_of_2_non_deleted] PASSED
e2e/moa/test_moa_entrypoints.py::test_regenesis_upgrade[6_of_3_deleted_1_2_6] PASSED
e2e/moa/test_moa_signer.py::test_moa_signer_valid[basic_stark_signer] PASSED
e2e/moa/test_moa_signer.py::test_moa_signer_valid[with_secp256r1_no_multisig] PASSED
e2e/moa/test_moa_signer.py::test_moa_signer_valid[with_secp256r1_multisig] PASSED
e2e/moa/test_moa_signer.py::test_moa_signer_valid[with_webauthn_no_multisig] PASSED
e2e/moa/test_moa_signer.py::test_moa_signer_valid[with_webauthn_multisig] PASSED
```

10 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.