Security Review
NM-0052: Argent Account on StarkNet

NETHERMIND

May 16, 2022

Contents

1 Executive Summary 2

2 Audited Files 3

3 Summary of Findings 4

4 Findings 5

5 Documentation Evaluation 9

6 Test Suite Evaluation 9

6.1 Tests Output for the Initial Report . . . 9

6.2 Tests Output for the Final Report . . . 10

7 About Nethermind 11

1 Executive Summary

This document presents the security review performed by Nethermind on the source code for the Argent Account on StarkNet. The project is composed of 4 (four) contracts (`ArgentAccount.cairo`, `Multicall.cairo`, `Proxy.cairo` and `Upgradable.cairo`), along with 2 (two) libraries (`AddressRegistry.cairo` and `ERC20.cairo`). The audit scope is restricted to the following files: `ArgentAccount.cairo`, `Proxy.cairo` and `Upgradable.cairo`. The project also contains a high level specification, which was referenced during the audit process. The Argent Account on StarkNet is 2-of-2 custom multisig where the `signer key` is typically stored on the user's phone and the `guardian key` is managed by an off-chain service to enable fraud monitoring. The guardian acts both as a co-validator for typical operations of the wallet, and as the trusted actor that can recover the wallet in case the `signer key` is lost or compromised. Alternatively, the user can add a second `backup_guardian key` to the account that has the same role as the `guardian` and can be used as the ultimate censorship resistance guarantee. The audit was carried out using manual inspection of the code base, interview with the client and automated tests provided by the Argent team. This document reports 8 (eight) points of attention. All the `high severity` issues were promptly addressed by the Argent team. **In summary, 4 (four) issues have been resolved and 4 (four) issues have been acknowledged by the client. The code is well-written, easy to follow and is actively under development by the Argent team**.

Code

Phase	Repository	Commit
Initial Report	Argent Account on StarkNet	7225b83f30
Final Report	Argent Account on StarkNet	PR#40

Summary of the Audit

Audit Type	Security Review
Initial Report	May 9, 2022
Final Report	May 16, 2022
Methods	Manual Review
Documentation	https://github.com/argentlabs/argent-contracts-starknetreadme

2 Audited Files

Cairo Files Reviewed (Initial Review)

File	md5sum hash
ArgentAccount.cairo	c847376ff551a6743569ec4962ce6ab0
Proxy.cairo	64ee566dcd609fbf01489f1425263793
Upgradable.cairo	b59137fdedc9235f820f4d3f85963218

Cairo Files Reviewed (Final Review)

File	md5sum hash
ArgentAccount.cairo	3258c26fe8ed3e9f326145a24075d0e5
Proxy.cairo	6c29c9f33aa2fc00669b098428836ccf
Upgradable.cairo	Same as initial review

Test Files (Initial Review)

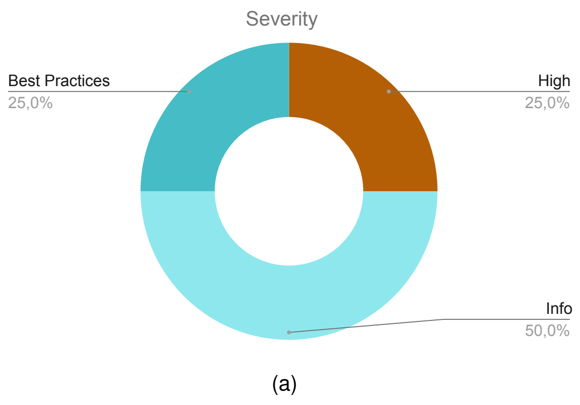
File	md5sum hash
address_registry.py	fef7f78a2b9a351c2d7e5c54f98f3335
argent_account.py	eef4f9238ab566897b61dd0b0fabfdc0
multicall.py	96616847021cfb076235215ef1fcabef
proxy.py	8e7a2d275f20c7c15e7b787ecd61584d
struct_hash.py	9f647df96c753daeb45d01a67a2451d8

Test Files (Final Review)

File	md5sum hash
address_registry.py	Same as initial review
argent_account.py	Same as initial review
multicall.py	Same as initial review
proxy.py	b63a3a9c68c6b5312689473777628581
struct_hash.py	Same as initial review

3 Summary of Findings

This audit report indicates 8 (eight) findings. The distribution of the findings according to the severity is presented in Fig. 1(a), while the distribution of findings according to the status of the issue is presented in Fig. 1(b). Each one of the findings is listed in the Table 'Summary of Findings'.



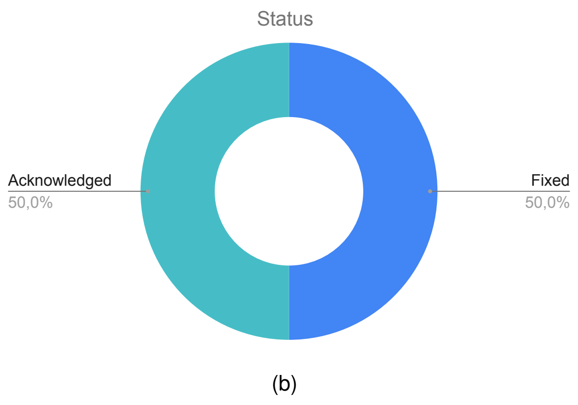


Figure 1: Fig. (a) presents the distribution of findings according to the severity. Fig. (b) presents the status of the findings.

Summary of Findings

Finding	**Severity**	**Update**
Contract can be interacted with before initialization	High	Fixed
The time-lock for escaping a guardian can be bypassed	High	Fixed
No event emitted during contract initialization	Info	Fixed
Batch of calls can fail	Info	Acknowledged
`Signer`, `Guardian`, and `Backup Guardian` can have the same address	Info	Acknowledged
Results from multiple calls are packed into a single array of bytes	Info	Acknowledged
State variable is overloaded to indicate if contract has been initialized	BP*	Acknowledged
Unused functions	BP*	Fixed

* BP = Best Practices

4 Findings

F01: [HIGH] Contract can be interacted with before initialization

Context: `ArgentAccount.cairo#L160-L235`

Description: The function `ArgentAccount.__execute__(...)` can be called before the contract has been initialized. Since all contract logic assumes that the function `ArgentAccount.initialize(...)` is called before any other function, the code must assert this intended behavior. Thus, the function `ArgentAccount.__execute__(...)` must perform a check to ensure that the contract has been properly initialized. Furthermore, the function `ArgentAccount.__execute__(...)` must require that `ArgentAccount._signer` is not `address(0x0)` and revert otherwise.

Recommendation:

1. Designate an specific state variable for recording if the contract has been initialized. Set this variable accordingly in the function `ArgentAccount.initialize(...)`. Whenever executing any action on the contract, firstly check if this state variable contains the proper value, especially in the core function `ArgentAccount.__execute__(...)`.
2. In the function `ArgentAccount.__execute__(...)` assure that `ArgentAccount._signer` is different from `address(0x0)`. Revert otherwise.

Argent Team Response: 1) added a `assert_initialized` check at the beginning of the `__execute__` method to make sure the signer is not 0; 2) updated the constructor of the proxy to execute a first `delegatecall` to the implementation. This makes it possible to create and initialize the account in one transaction.

F02: [HIGH] The time-lock for escaping a guardian can be bypassed

Context: `ArgentAccount.cairo#L399-L421`

Description: The time-lock of seven days to call `escape_guardian(...)` can be bypassed by the `signer` by calling `escape_guardian(...)` when the storage variable `_escape`'s struct has both values `active_at` and `type` equal to zero. It is possible for `_escape` to be in this state when there is no current escape in the contract. The function `escape_guardian(...)` has two checks to ensure that the escape is valid: a) the current time has exceeded the `current_escape.active_at`, i.e., enough time has passed since `trigger_escape_guardian` has been called; b) the escape type is `ESCAPE_TYPE_GUARDIAN`. When there is no current escape in the contract both values are zero so the checks pass in the following way:

```
assert_le(current_escape.active_at, block_timestamp)
# Variable `current_escape.active_at` is zero.
# Variable `block_timestamp` is Unix time and will always be above zero.
# This assert statement will pass.
assert current_escape.type = ESCAPE_TYPE_GUARDIAN
#Variable `current_escape.type` is zero.
#Constant `ESCAPE_TYPE_GUARDIAN` is zero.
#This assert statement will pass.
```

Both asserts will pass and `_guardian` will be changed immediately instead of having to wait seven days which is the expected functionality. The following test can be added to `test/argent_account.py` to verify this:

```python
@pytest.mark.asyncio
async def test_bypass_escape_guardian_timelock(get_starknet, account_factory):
    account = account_factory
    starknet = get_starknet
    sender = TransactionSender(account)
    new_guardian = Signer(55555555)
    # reset block_timestamp
    reset_starknet_block(starknet=starknet)
    # trigger immediate escape without calling `trigger_escape_guardian`
    tx_exec_info = await sender.send_transaction(
        [
            (
                account.contract_address,
                'escape_guardian',
                [new_guardian.public_key]
            )
        ],
        [
            signer
        ]
    )
    assert_event_emmited( tx_exec_info, from_address=account.contract_address, name='guardian_escaped',
    ↪  data=[new_guardian.public_key] )

    assert (await account.get_guardian().call()).result.guardian == (new_guardian.public_key)
```

Recommendations: To address this issue both of the following recommended changes should be implemented.

1. Change the constants `ESCAPE_TYPE_GUARDIAN` and `ESCAPE_TYPE_SIGNER` so that neither constants are zero.
2. Add a sanity check to ensure that `current_escape.active_at` is not zero in the functions `escape_guardian` and `escape_signer`.

Argent Team Response: 1) changed the values of `ESCAPE_TYPE_GUARDIAN` and `ESCAPE_TYPE_SIGNER` to be non-zero; 2) added the sanity check `assert_not_zero(current_escape.active_at)` in `escape_guardian` and `escape_signer`.

F03: [INFO] No event emitted during contract initialization

Context: `ArgentAccount.cairo#L145-L167`

Description: The function `ArgentAccount.initialize(...)` makes changes to state but does not emit any event. This function is the basis from which the code is built upon, and the emission of an event with the input parameters could support post-deployment investigations.

Recommendation: Emit an event in the last line of `ArgentAccount.initialize(...)` containing `_signer` and the `_guardian`.

Argent Team Response: Added the `account_initialized` event.

F04: [INFO] Batch of calls can fail

Context: `ArgentAccount.cairo#L169-L235`

Description: In the case where you call the function `ArgentAccount.__execute__(...)` to run a batch of calls at once, should any of them fail this will cause the entire transaction to revert. StarkNet does not support the ability to "catch" failed asserts so it is not possible to allow all other successful calls to run. This should be explained to users as there can be circumstances where the user may want all other calls to succeed. For example: A batch of calls to swap different tokens on an exchange, if one swap reverts (EG: a swap exceeds its defined slippage) then the entire transaction will fail including all other swap calls that would have otherwise succeeded.

Recommendation: Clearly state this condition in the user facing documentation.

Argent Team Response: Will make sure this is properly documented in the future.

F05: [INFO] `Signer`, `Guardian`, and `Backup Guardian` can have the same address

Context: `ArgentAccount.cairo#L125-L135`

Description: By analyzing the functions `ArgentAccount.change_signer(...)`, `ArgentAccount.change_guardian(...)`, and `ArgentAccount.change_guardian_backup(...)` we notice that the same address can be set to the state variables `ArgentAccount._signer`, `ArgentAccount._backup`, and `ArgentAccount._guardian_backup`. As explained by the Argent team, this is the intended behavior.

Recommendation: Since this is the intended behavior, no action is required from the client.

Argent Team Response: As discussed this is a conscious choice but will make sure it is properly documented.

F06: [INFO] Results from multiple calls are packed into a single array of bytes

Context: `ArgentAccount.cairo#L688-L748`

Description: The function `ArgentAccount.execute_list(...)` is responsible for executing an array of actions. The output of these actions are placed into the variable `execute_list.reponse` (notice the missing `s` in the variable name) without including the size of each particular response. The developers are already aware of this situation.

Recommendation: The decision on how the system must operate belongs to the client. Thus, no action is required from the client on this matter.

Argent Team Response: As discussed this is intentional and the unpacking is left to the calling contract.

F07: [BP] State variable is overloaded to indicate if contract has been initialized

Context: `ArgentAccount.cairo#L145-L167`

Description: The state variable `ArgentAccount._signer` in function `ArgentAccount.initialize(...)` is overloaded. Its purposes are to keep track of the signer as well as signal if the contract has already been initialized. In some case where `ArgentAccount._signer` is reset to `address(0x0)`, both of these purposes will be affected and anybody would be able to call `ArgentAccount.initialize(...)`. The current implementation of the `ArgentAccount` contract prevents `_signer` from being reset, however splitting this overloaded state variable into two state variables with specific purposes could make the code more robust.

Recommendation: Consider designating a specific state variable for recording if the contract has been initialized or ensure that no future functionality will be able to reset `ArgentAccount._signer` to `address(0x0)`.

Argent Team Response: This is a conscious choice to lower the cost of creating accounts as on StarkNet the main cost comes from storage vars that need to be propagated to L1. We do not see a need to introduce another storage var given that the contract logic guarantees the account is initialized if and only if the signer is not 0.

F08: [BP] Unused functions

Context: `Proxy.cairo`, `ArgentAccount.cairo`

Description: In `Proxy.cairo` the imported function `assert_not_zero(...)` is never used. In `ArgentAccount.Cairo` the imported functions `get_fp_and_pc()`, `hash_init(...)`, `hash_finalize(...)`, `hash_update(...)`, and `hash_update_single(...)` are never used.

Recommendation: Remove unused imports and functions from the code to improve overall readability and maintainability.

Argent Team Response: Unused methods removed.

5 Documentation Evaluation

The volume of inline comments can be improved. In its actual form, the audited files present 709 lines of code and just 108 lines of comments. As the system grows and novel functionalities are included, having a robust development documentation can make a huge difference. Thus, we would like to recommend formalizing the functional requirements of this project, keeping this updated while the project is being built.

The `README.md` is missing the information for compiling `ArgentAccount.cairo`. It would be helpful to state that the compile order should be:

1. `nile compile`
2. `nile compile --account_contract contracts/ArgentAccount.cairo`

6 Test Suite Evaluation

The test suite has room for improvement. In its actual format, it performs the basic validations required for unitary tests. However, systemic validations can be incorporated in order to catch non-trivial edge cases. The test suite is composed of the following files:

– `contracts/test/StructHash.cairo`
– `contracts/test/TestDapp.cairo`

In the following subsections, we present the tests output for the initial report and the final report.

6.1 Tests Output for the Initial Report

```
pytest ./test/argent_account.py
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
configfile: pyproject.toml
plugins: asyncio-0.18.3, web3-5.29.0, typeguard-2.13.3
asyncio: mode=legacy
collected 17 items

test/argent_account.py::test_initializer PASSED
test/argent_account.py::test_call_dapp_with_guardian PASSED
test/argent_account.py::test_call_dapp_no_guardian PASSED
test/argent_account.py::test_multicall PASSED
test/argent_account.py::test_change_signer PASSED
test/argent_account.py::test_change_guardian PASSED
test/argent_account.py::test_change_guardian_backup PASSED
test/argent_account.py::test_change_guardian_backup_when_no_guardian PASSED
test/argent_account.py::test_trigger_escape_guardian_by_signer PASSED
test/argent_account.py::test_trigger_escape_signer_by_guardian PASSED
test/argent_account.py::test_trigger_escape_signer_by_guardian_backup PASSED
test/argent_account.py::test_escape_guardian PASSED
test/argent_account.py::test_escape_signer PASSED
test/argent_account.py::test_signer_overrides_trigger_escape_signer PASSED
test/argent_account.py::test_guardian_overrides_trigger_escape_guardian PASSED
test/argent_account.py::test_cancel_escape PASSED
test/argent_account.py::test_is_valid_signature PASSED
== 17 passed in 294.56s (0:04:54) ==
```

6.2 Tests Output for the Final Report

```
platform linux -- Python 3.8.10, pytest-7.1.2, pluggy-1.0.0
configfile: pyproject.toml
plugins: asyncio-0.18.3, typeguard-2.13.3, web3-5.29.0
asyncio: mode=legacy
collected 17 items

test/argent_account.py::test_initializer PASSED
test/argent_account.py::test_call_dapp_with_guardian PASSED
test/argent_account.py::test_call_dapp_no_guardian PASSED
test/argent_account.py::test_multicall PASSED
test/argent_account.py::test_change_signer PASSED
test/argent_account.py::test_change_guardian PASSED
test/argent_account.py::test_change_guardian_backup PASSED
test/argent_account.py::test_change_guardian_backup_when_no_guardian PASSED
test/argent_account.py::test_trigger_escape_guardian_by_signer PASSED
test/argent_account.py::test_trigger_escape_signer_by_guardian PASSED
test/argent_account.py::test_trigger_escape_signer_by_guardian_backup PASSED
test/argent_account.py::test_escape_guardian PASSED
test/argent_account.py::test_escape_signer PASSED
test/argent_account.py::test_signer_overrides_trigger_escape_signer PASSED
test/argent_account.py::test_guardian_overrides_trigger_escape_guardian PASSED
test/argent_account.py::test_cancel_escape PASSED
test/argent_account.py::test_is_valid_signature PASSED
== 17 passed in 337.83s (0:05:37) ==
```

7 About Nethermind

Founded in 2017 by a small team of world-class technologists, Nethermind builds Ethereum solutions for developers and enterprises. Boosted by a grant from the Ethereum Foundation in August 2018, our team has worked tirelessly to deliver the fastest Ethereum client in the market. Our flagship Ethereum client is all about performance and flexibility. Built on .NET core, a widespread, enterprise-friendly platform, Nethermind makes integration with existing infrastructures simple, without losing sight of stability, reliability, data integrity, and security

Nethermind is made up of several engineering teams across various disciplines, all collaborating to realize the Ethereum roadmap, by conducting research and building high-quality tools. Teams focus on specific areas of the Ethereum problem space. Each consists of specialists and experienced developers working alongside interns, learning the ropes in the Nethermind Internship Program.

Our mission is to gather passionate talent from around the world, and to tackle some of the blockchain's most complex problems. Nethermind provides software solutions and services for developers and enterprises building the Ethereum ecosystem. We offer security reviews to projects built on EVM compatible chains and StarkNet. We have expertise in multiple areas of the Ethereum ecosystem, including protocol design, smart contracts (written in Solidity and Cairo), MEV, etc. We develop some of the most used tools on Starknet and one of the most used Ethereum clients. Learn more about us at `https://nethermind.io`.