Argent Account & Multisig for StarkNet

1 Executive Summary

2 Scope

2.1 Objectives

3 System Overview

3.1 Argent Account

3.2 Argent Multisig

3.3 Overall

4 Security Specification

4.1 Actors

4.2 Trust Model

5 Findings

5.1 Inconsistency Between Actual `IAccount` Interface and Published Interface ID Medium ✓ Fixed

5.2 Wrong ID for `OutsideExecution` Interface Medium ✓ Fixed

5.3 `change_owner` Selector Test Missing ✓ Fixed

5.4 Suggestions for Minor Code Quality Improvements ✓ Fixed

6 Recommendations

6.1 Hard-Coded Grace Period Lengths for Escapes Might Not Suit Every User

6.2 If the Guardian and Backup Guardian Are Both Compromised, the Escape Gas Attack Spam Can Be Repeated

Appendix 1 - Files in Scope

Appendix 2 - Disclosure

Date	June 2023
Auditors	George Kobakhidze, Heiko Fisch

1 Executive Summary

This report presents the results of our engagement with **Argent** to review **Argent Account** and **Argent Multisig** for StarkNet.

The review was conducted from **May 29** to **June 30, 2023**, by **Heiko Fisch** and **George Kobakhidze**. A total of 8 person-weeks were spent.

Argent is a provider of wallets in the Ethereum ecosystem, including its L2s. This audit focuses on their smart contract wallet offering, Argent Account, and their multisig, Argent Multisig, as well as the auxiliary libraries used for these. Specifically, the code to be reviewed is for the StarkNet ecosystem, which is currently undergoing a lot of fundamental changes due to the release of Cairo 1.0 and compatible StarkNet contracts. Therefore, these are StarkNet contracts written in the Cairo 1.0 programming language.

We found the code exceptionally well-written, -documented, and extensively tested. As the StarkNet/Cairo ecosystem is evolving rapidly and Argent is trying to keep up with these changes, we accepted several code updates during the engagement. The findings described below refer to different versions of the codebase. They are all fixed in the final version `d5b365b93f7342421cd4e7775998c91acb2d1a20`.

Users of Argent Account should be aware that the time window to react to a malicious Guardian's takeover attempt is only one week and should take appropriate precautions.

2 Scope

Our review initially focused on the commit hash `5be99f790267108e315022b60b00d9608093f514`. Due to the changing state of the Cairo 1.0 programming language, there were several PRs and commits incorporated into this audit with the major ones being:

1. PR #159 to address a non-whitelisted function.
2. PR #160 to upgrade to Cairo v1.1.0.
3. PR #174 to be compliant with SNIP-5 spec.
4. PR #175 to migrate to the syntax required by the next Cairo compiler version (v2.0.0) that will become available on Starknet v0.12.0.
5. PR #182 to switch to the latest version of the `IAccount` interface.

After addressing found issues, the final commit for the repository was `d5b365b93f7342421cd4e7775998c91acb2d1a20`.

The list of files in scope can be found in the Appendix with file hashes related to the latest commit.

Backwards-compatibility with the Cairo 0 versions was not in scope for this engagement.

2.1 Objectives

Together with the **Argent** team, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry as they would translate to Cairo 1.0 StarkNet contracts.
3. Address key risks:

- Assets (ERC20, ERC721, etc) owned by the account cannot be stolen or frozen by a malicious party.
- The account can never be in a bricked state.
- The impact of a malicious guardian is limited.

4. Verify invariants:

- The account has always an owner (`signer` is never 0).
- The `guardian` key can never be `0` when the `guardian_backup` is set.

3 System Overview

The two systems that this audit was focused on – Argent Account and Argent Multisig – can be assessed in isolation from one another. While their primary use case of providing a self-custodial smart contract wallet solution is shared, the details and target users are different.

3.1 Argent Account

Argent Account is a smart contract wallet intended to be used by individuals. This wallet allows users to sign their everyday transactions for things like asset transfers, smart contract interactions and so on. There is also a new concept of a Guardian, an optional entity that helps users increase the security and recoverability of their wallet. To achieve the increased security goals, Argent Accounts employ a few more layers of logical separation than what is usually found in wallets for transaction signing, ownership, and signer hierarchy.

To execute any transaction on behalf of the Argent Account, the account checks to see if enough valid signatures are present with the transaction. This always includes the owner signature, which is the signature created from the private key that corresponds to the address referred to as the owner in the Argent Account, and could also require a Guardian or the backup Guardian signature as well. In fact, if a Guardian is defined, most transactions on behalf of the Argent Account would require for the Guardian signature to be there. Guardians are third parties typically in the form of an off-chain monitoring service with a private key that would check if the transaction submitted by the user looks suspicious, and, in the case that it is, wouldn't sign the transaction, preventing it from being executed on behalf of the Argent Account. This is done to offer a fail-safe layer to the users so they don't accidentally sign something malicious or, for example, provide a monetary input greater than what the user has configured with the Guardian. There is also the concept of a "Backup Guardian", which would have exactly the same privileges as the primary Guardian and exists to provide greater service availability to the user in case the first Guardian goes offline. It is important to note that most transactions need to be signed by both the owner and the Guardian, assuming the latter is defined; neither can execute arbitrary transactions alone. The exceptions would be the abilities to escape (and disable) the Guardian by the owner and escape the owner by the Guardian. It is critical to note that in these escape scenarios, the owner takes precedence. More on that below.

The ownership of the account itself in Argent Accounts is transferrable. If a user would like to change the private keys that have owner access to the Argent Account, they may do so by calling the `change_owner` function on their Argent wallet. This, however, requires a signature of the initial owner itself, which could be a problem if the ownership transfer needs to happen because the original private key is lost. However, the aforementioned Guardian entity, should it be enabled and defined, may also facilitate ownership transfer by itself via the `escape_owner` method. This functionality initiates a process, with the current length set to 7 days, at the end of which the ownership of the account may be transferred to the account the Guardian has defined. Should the original owner keys be found, this process can be overridden at any point by the original owner. In fact, the owner themselves may also call the `change_guardian` function to change or disable the Guardian, if they feel like they don't need or want the Guardian service, though this would require the sign-off of the Guardian too. If the Guardian is compromised, meaning it wouldn't sign its ownership away willingly, the owner may initiate the `escape_guardian` process, which acts similarly as the owner escape except that the Guardian can't override or cancel this escape.

Finally, since Argent Accounts are smart contract wallets, they also allow for outside services to process transactions on behalf of the account should they have the necessary signatures. This is done through the `execute_from_outside` method. A feature like this would enable use cases such as sponsored transactions, gas subsidized UX and so on.

By spreading the ability to optionally sign and check transactions with multiple entities with different signing powers, the Argent Accounts allow for a safer self-custody experience than regular wallets. However, there are additional trust assumptions with respect to the Guardians and the escape period (currently a week) that do require strong consideration from users. For example, the user would need to make sure they monitor their wallet and its transactions to make sure that in case a Guardian is compromised and tries to escape the owner, they act swiftly within the security period and cancel the malicious ownership escape.

3.2 Argent Multisig

The Argent Multisig provides a typical feature implementation in Cairo 1.0 of an account that requires M-N signatures to process a transaction.

The contract contains typical multisig management functionality such as adding signers, removing signers, replacing one signer with another, changing thresholds for signature processing and so on. All the necessary invariants are enforced, such as ensuring that a `0` signer can't be added and that the threshold isn't greater than the total number of signers available. For efficiency purposes, the number of signers that can be registered is limited to 32, so users should be aware of this limit.

Like with the Argent Account, the Argent Multisig also allows third parties to execute transactions on behalf of the multisig via the `execute_from_outside` method, provided the third party has the necessary signatures.

3.3 Overall

Both of the above contracts utilize `check_ecdsa_signature` from the Cairo `corelib::ecdsa` library to verify the signatures against provided hashes. They do not utilize smart contract signatures at this moment.

Finally, both contract systems may be upgraded by their relevant signatory entities (owners and Guardians for the Argent Account, and signers for the Argent Multisig). The upgrade logic is utilizing Cairo 1.0 methodology with the new `replace_class_syscall` function along with custom logic as needed for each contract, such as verifying that the new class (implementation contract) supports relevant interfaces.

4 Security Specification

This section describes, from a security perspective, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are listed below with their respective abilities:

- Argent Account – Owners. The primary user of the accounts. In the vast majority of cases, they initiate and sign off on the transactions. They also hold the most power over the account; should their access be compromised by a malicious actor, the account would be in trouble. In the case of a loss of access, Guardians may help.
- Argent Account – Guardians & Backup Guardians. Third parties that allow for fail-safe mechanisms for the Argent Account. These entities are optionally setup in Argent Accounts, and may be used to filter and sign off on all the transactions made by the Argent Account, as well as initiate ownership transfers if private keys to the primary owner are lost.

- Argent Multisig – Signers. The primary and only users of the Argent Multisig. These entities all hold the same power on the multisig, as defined throughout the multisig's lifecycle. At least N signatures (the threshold) of M total number of signers is required to process any transaction on behalf of the Argent Multisig.

4.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- Argent Account – Owners. As mentioned above, the owners are the primary users of Argent Accounts. As the name suggests, as the owner they have full control over the account. So, if the users have access to the private keys associated with the owner address, they don't need to extend any trust assumptions for ownership.

- Argent Account – Guardians & Backup Guardians. Despite offering a safety layer to the Argent Accounts, Guardians and Backup Guardians are trusted to perform their job and not act maliciously, though the latter actions are quite limited. The Guardians do need to perform proper monitoring of transactions and act in accordance with the user agreement that they have made with the owner. For example, that could be checking for appropriate slippage limits set on swap transactions, ensuring daily limits of asset transfers aren't exceeded and so on. On top of monitoring, the Guardians need to make sure they are available to sign off on transactions as well, otherwise the Argent Account wouldn't be able to validate any transactions, and the owner would have to initiate a Guardian escape that would take a week.
  Finally, the Guardians are trusted not to be compromised by a malicious actor. If they are, they can perform a gas attack by spamming ownership escape. In StarkNet, gas for transactions is spent from the smart contract accounts that may have various ownership properties as opposed to public addresses directly derived from private keys. Therefore, a private key that has no coins for gas in their public address but is a signatory on a smart contract account may sign and execute a transaction that will spend gas costs from the account – in this case the Argent Account. Since it is possible for Guardians to initiate the escape owner transactions to start the escape process, they could spam this same transaction and waste assets stored in this account for gas. The damage would be limited, though, as the contract has limits on the maximum gas fee for escape transactions and on the number of escape attempts possible by Guardians. However, for a malicious Guardian only one ownership escape needs to go through to take over the account, and this would only take a week. If the owner falls ill or doesn't have access to the internet or their account for just a week, a malicious Guardian could seize the account.

- Argent Multisig – Signers. Argent Multisig signers are the primary and only users of the multisig. All transactions require at least a threshold amount of signatures for them to be processed. As a result, all signers should be aware and trust at least `M (total number of signers) - N (threshold of signers) + 1` other signers associated with the multisig to not sign malicious transactions.

5 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Inconsistency Between Actual `IAccount` Interface and Published Interface ID Medium ✓ Fixed

Resolution

After the official end of this engagement, the community has come to the decision that `is_valid_signature` should return the StarkNet constant `VALIDATED` (which is a `felt252` that represents the string `'VALID'`) in the affirmative case and `0` otherwise. Therefore, the interface ID to be returned is now set to be `0x2ceccef7f994940b3962a6c67e0ba4fcd37df7d131417c604f91e03caecc1cd`. The Argent team has changed the implementation accordingly and provided us with the updated commit hash. We have reviewed the changes and updated the final commit hash of the report to this new version.

Description

As an analog to Ethereum Improvement Proposals (EIPs), there are StarkNet Improvement Proposals (SNIPs), and SNIP-5 – similar in intention and technique to ERC-165 – defines how to publish and detect what interfaces a smart contract implements. As in ERC-165, this is achieved with the help of an *interface identifier*.

Specifically, this ID is defined as the XOR of the "extended function selectors" in the interface. While not going into all the details here, a function's extended selector is the `starknet_keccak` hash of its signature, where some special rules define how to deal with the different data types. The details can be found in the proposal. Compliant contracts implement a `supports_interface` function that takes a `felt252` and returns `true` if the contract implements the interface with this ID and `false` otherwise. For example, `argent_account.cairo` defines the following `supports_interface` function:

contracts/account/src/argent_account.cairo:L506-L522

```
fn supports_interface(self: @ContractState, interface_id: felt252) -> bool {
    if interface_id == ERC165_IERC165_INTERFACE_ID {
        true
    } else if interface_id == ERC165_ACCOUNT_INTERFACE_ID {
        true
    } else if interface_id == ERC165_OUTSIDE_EXECUTION_INTERFACE_ID {
        true
    } else if interface_id == ERC165_IERC165_INTERFACE_ID_OLD {
        true
    } else if interface_id == ERC165_ACCOUNT_INTERFACE_ID_OLD_1 {
        true
    } else if interface_id == ERC165_ACCOUNT_INTERFACE_ID_OLD_2 {
        true
    } else {
        false
    }
}
```

In this issue, we're interested in `ERC165_ACCOUNT_INTERFACE_ID`, which is defined as follows:

contracts/lib/src/account.cairo:L3-L4

```
const ERC165_ACCOUNT_INTERFACE_ID: felt252 =
    0x32a450d0828523e159d5faa1f8bc3c94c05c819aeb09ec5527cd8795b5b5067;
```

This ID corresponds to an interface with the following function signatures:

```
    fn __validate__(Array<Call>) -> felt252;
    fn __execute__(Array<Call>) -> Array<Span<felt252>>;
    fn is_valid_signature(felt252, Array<felt252>) -> bool;
```

Note that `is_valid_signature` returns a `bool`. However, in the actual `IAccount` interface, this function returns a `felt252`:

contracts/lib/src/account.cairo:L10-L17

```
// InterfaceID: 0x32a450d0828523e159d5faa1f8bc3c94c05c819aeb09ec5527cd8795b5b5067
trait IAccount<TContractState> {
    fn __validate__(ref self: TContractState, calls: Array<Call>) -> felt252;
    fn __execute__(ref self: TContractState, calls: Array<Call>) -> Array<Span<felt252>>;
    fn is_valid_signature(
        self: @TContractState, hash: felt252, signatures: Array<felt252>
    ) -> felt252;
}
```

If we check out the implementation of `is_valid_signature`, we see that it returns the magic value `0x1626ba7e` known from ERC-1271 if the signature is valid and `0` otherwise:

contracts/account/src/argent_account.cairo:L214-L222

```
fn is_valid_signature(
    self: @ContractState, hash: felt252, signatures: Array<felt252>
) -> felt252 {
    if self.is_valid_span_signature(hash, signatures.span()) {
        ERC1271_VALIDATED
    } else {
        0
    }
}
```

contracts/lib/src/account.cairo:L8

```
const ERC1271_VALIDATED: felt252 = 0x1626ba7e;
```

The ID for this interface would be `0x2ceccef7f994940b3962a6c67e0ba4fcd37df7d131417c604f91e03caecc1cd`. Note that, unlike in ERC-165, in SNIP-5, the return type of a function does matter for the interface identifier. Hence, the actual `IAccount` interface defined and implemented and the published interface ID do not match.

Recommendation

At the end of this engagement, the community has not come to a decision yet whether `is_valid_signature` should return a `bool` or a `felt252` (i.e., the magic value `0x1626ba7e` in the affirmative case and `0` otherwise). Depending on the outcome, either the actual interface and its implementation or the published interface ID must be changed to achieve consistency between the two.

Remark

SNIP-5 is not very clear on how to deal with the new Cairo syntax introduced in v2.0.0 of the compiler. Specifically, with this new syntax, interface traits have a generic parameter `TContractState`, and all non-static functions in the interface have a first parameter `self` of type `TContractState` or `@TContractState` for `view` functions. How to deal with this parameter in the derivation of the interface identifier is not (yet) explicitly specified in the proposal, but the Argent team has assured us that the understanding in the community is to ignore this parameter for the extended function selectors and hence the interface ID.

5.2 Wrong ID for `OutsideExecution` Interface Medium ✓ Fixed

Description

While not standardized across the community, the Argent team has decided to isolate the "outside execution" functionality in a separate interface, so other teams in the ecosystem can choose to implement that interface as well.

contracts/lib/src/outside_execution.cairo:L10-L29

```
/// Interface ID: 0x3a8eb057036a72671e68e4bad061bbf5740d19351298b5e2960d72d76d34cb9
// get_outside_execution_message_hash is not part of the standard interface
#[starknet::interface]
trait IOutsideExecution<TContractState> {
    /// @notice This method allows anyone to submit a transaction on behalf of the account as long as they have the relevant signat
    /// @param outside_execution The parameters of the transaction to execute
    /// @param signature A valid signature on the Eip712 message encoding of `outside_execution`
    /// @notice This method allows reentrancy. A call to `__execute__` or `execute_from_outside` can trigger another nested transac
    fn execute_from_outside(
        ref self: TContractState, outside_execution: OutsideExecution, signature: Array<felt252>
    ) -> Array<Span<felt252>>;

    /// Get the status of a given nonce, true if the nonce is available to use
    fn is_valid_outside_execution_nonce(self: @TContractState, nonce: felt252) -> bool;

    /// Get the message hash for some `OutsideExecution` following Eip712. Can be used to know what needs to be signed
    fn get_outside_execution_message_hash(
        self: @TContractState, outside_execution: OutsideExecution
    ) -> felt252;
}
```

SNIP-5 – as already mentioned in issue 5.1 – is a StarkNet Improvement Proposal that describes how to publish and detect what interfaces a contract implements. To briefly summarize, the interface ID is defined as the XOR of the extended selectors of the functions in the interface, and a function's extended selector is the `starknet_keccak` hash of the function signature, where some special rules define how to deal with the different data types. Deriving the input for `starknet_keccak` can be done manually, but it is tedious, error-prone, and can even be somewhat involved, as it may require knowledge of some Cairo internals, depending on the types used in the function.

When we tried to verify the ID for the `OutsideExecution` interface, we noticed a mismatch between the result of our own calculations and the ID the Argent team had arrived at:

contracts/lib/src/outside_execution.cairo:L7-L8

```
const ERC165_OUTSIDE_EXECUTION_INTERFACE_ID: felt252 =
    0x3a8eb057036a72671e68e4bad061bbf5740d19351298b5e2960d72d76d34cb9;
```

Together with the client, we were able to identify a mistake that was made in the manual derivation of the input to the hash function, leading to a wrong extended function selector and, therefore, an incorrect interface identifier.

The correct extended function selector for `execute_from_outside` is:

```
starknet_keccak(
    'execute_from_outside(
        (ContractAddress,felt252,u64,u64,(@Array<(ContractAddress,felt252,Array<felt252>)>)),
        Array<felt252>
    )->Array<(@Array<felt252>)>'
) = 0x3c6e798a947887809ab7c506818dac2e3632acafa20cb51d2fff56b3577dc75
```

(The line breaks were only inserted for better readability in this document. The string does not contain any whitespace.)

Recommendation

Together with the extended function selector for `is_valid_outside_execution_nonce`, `0x3ae284922d559e87220df9c5a51dae59c391ce8f3b4fabb572275e210299df4`, the resulting interface ID for `OutsideExecution` is `0x68cfd18b92d1907b8ba3cc324900277f5a3622099431ea85dd8089255e4181`, and the definition of `ERC165_OUTSIDE_EXECUTION_INTERFACE_ID` should be changed accordingly.

Note that the Argent team has deliberately omitted `get_outside_execution_message_hash` from the interface (in the sense of SNIP-5).

5.3 `change_owner` Selector Test Missing ✓ Fixed

Description

The Argent Account contract employs many hard-coded constants in its logic, for example for the function selectors. Consequently, there is a test for each one of these selectors. Although the tests were not in scope for this audit, we noticed that one selector test is missing – the `change_owner` selector with value `0x580363632898419625012472292490227837275277578340436814344857564692360760 08`.

contracts/account/src/argent_account.cairo:L46-L47

```
const CHANGE_OWNER_SELECTOR: felt252 =
    0x580363632898419625012472292490227837275277578340436814344857564692360760 08; // starknet_keccak('change_owner')
```

contracts/account/src/tests/test_argent_account.cairo:L222

```
fn test_selectors() {
```

Recommendation

Add the test for the `change_owner` selector.

5.4 Suggestions for Minor Code Quality Improvements ✓ Fixed

Descriptions and Recommendations

A. There are two opportunities for minor code quality improvements in the `execute_multicall` function:

contracts/lib/src/calls.cairo:L12-L39

```
fn execute_multicall(calls: Span<Call>) -> Array<Span<felt252>> {
    let mut result: Array<Span<felt252>> = ArrayTrait::new();
    let mut calls = calls;
    let mut idx = 0;
    loop {
        match calls.pop_front() {
            Option::Some(call) => {
                match call_contract_syscall(*call.to, *call.selector, call.calldata.span()) {
                    Result::Ok(mut retdata) => {
                        result.append(retdata);
                        idx = idx + 1;
                    },
                    Result::Err(revert_reason) => {
                        let mut data = ArrayTrait::new();
                        data.append('argent/multicall-failed');
                        data.append(idx);
                        data.append_all(revert_reason);
                        panic(data);
                    },
                }
            },
            Option::None(_) => {
                break ();
            },
        };
    };
    result
}
```

1. `retdata` is never changed and doesn't have be mutable.
2. Instead of using an immutable `calls` parameter for the function that is later shadowed by a mutable variable of the same name, it would make more sense to make the function parameter mutable right away.

B. The `append_all` function appends the elements of one array to another:

contracts/lib/src/array_ext.cairo:L8-L19

```
fn append_all(ref self: Array<T>, mut value: Array<T>) {
    loop {
        match value.pop_front() {
            Option::Some(reason) => {
                self.append(reason);
            },
            Option::None(()) => {
                break ();
            },
        };
    };
}
```

This code is currently only used in one place: in `execute_multicall` to append the return data of an erroneous call – the "revert reasons" – to another array. This probably explains why, in `append_all`, the variable that holds the individual elements of the array is named `reason`. Presumably the code that forms the body of `append_all` was used directly in `execute_multicall` – where the name `reason` was justified – and later moved to a separate `append_all` function. This makes sense, but in the general context, the name `reason` is unmotivated, and a better name would be `element` or something similar.

6 Recommendations

6.1 Hard-Coded Grace Period Lengths for Escapes Might Not Suit Every User

Description

Owner and Guardian can each trigger an escape of the other party. Before the escape can be activated, there is a grace period during which it can be canceled (if both parties agree) or during which the owner can unilaterally override an owner escape (triggered by the Guardian) with a Guardian escape. The length of this grace period is hard-coded to 7 days for both types of escape.

Obviously, there is a trade-off for the length of the grace periods:

1. Owner escape: If the owner has lost their private key and asks the Guardian to move the Account to a new owner, they'd want that to happen as quickly as possible to regain control of the account. On the other hand, if the Guardian is malicious and wants to replace the owner without the latter's consent, the owner would like to have as much time as possible to react and override this takeover attempt.
2. Guardian escape: If the Guardian becomes malicious or inactive, the owner would like to replace or remove this Guardian as fast as possible. On the other hand, if the owner's private key gets compromised, the Guardian might prevent the worst (e.g., via transfer limits) even before the owner has knowledge of the compromise and can work with the Guardian to move the Account to a new owner. Therefore, in this scenario, actually replacing or removing the Guardian should be delayed as long as possible.

Depending on factors such as trust in the Guardian and own availability/attention, different users of the Argent Account might have different preferences with regard to these trade-offs.

Recommendation

Even taking into account that an explicit design goal is simplicity, configurable grace period lengths would make Argent Account more adaptable to different types of users and different usage scenarios. In any case, the default setting and its implications should be clearly communicated to users.

6.2 If the Guardian and Backup Guardian Are Both Compromised, the Escape Gas Attack Spam Can Be Repeated

Description

The Argent Account has optional entities called Guardian and Backup Guardian that act as a fail-safe layer in monitoring and signing of transactions and in recovery of access to the account in the event of loss of keys to the owner.

To achieve the latter use case, the Guardians can execute an `escape_owner` process which initiates a security period (currently 7 days) after which the ownership of the account will go to a new owner.

contracts/account/src/argent_account.cairo:L294

```
fn trigger_escape_owner(new_owner: felt252) {
```

However, this method simply starts the process and can be restarted many times (even if the original process doesn't finish) as long as it isn't overridden by the owner with an `escape_guardian` process, which achieves the same outcome except the change will affect the Guardian instead of the owner. And since in StarkNet the gas costs for processing the transaction are paid for by the account, in this case our Argent Account, the `escape_owner` function can be spammed to waste coins from the account.

In particular, if the Guardian and/or the Backup Guardian are compromised, they could do this for malicious reasons. Thankfully, this is limited by counting the amount of attempts to start the escape process. In particular, this is currently limited to just 5 attempts. Similarly, there is a limit on the maximum fee as well:

contracts/account/src/argent_account.cairo:L49-L52

```
/// Limit escape attempts by only one party
const MAX_ESCAPE_ATTEMPTS: u32 = 5;
/// Limits fee in escapes
const MAX_ESCAPE_MAX_FEE: u128 = 50000000000000000; // 0.05 ETH
```

As a result, the damage is limited. Once the owner notices this, they can simply replace the Guardian through an escape with either a good actor or disable it altogether.

However, when the Guardian is replaced, the number of escape attempts is reset to 0. So if there is also a malicious Backup Guardian, they could start the gas attack again. The owner would then have to race the Backup Guardian to change/remove them as well before they start the gas attack to avoid any additional damage.

It is important to note that this would still be limited to just another `MAX_ESCAPE_ATTEMPTS` (currently 5) worth of escape transactions with `MAX_ESCAPE_MAX_FEE` (currently `5e16` of the smallest decimal of the gas coin, for example `0.05 ETH` for ether) , and the Backup Guardian won't be able to repeat this process, so the damage is again minimized.

Recommendation

In a conversation with the Argent team, this has been acknowledged as a design choice and, due to its minimized impact, will remain. It might be beneficial to describe this scenario in the docs when describing the possible gas attack by the malicious Guardian so the users are aware, especially for those with smaller wallets.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/account/src/argent_account.cairo	`df50077dbc2c2928c47fbc602c6bd70961fffb3f`
contracts/account/src/escape.cairo	`b30258ca6e74644391e193bda26b4aa3a125d538`
contracts/account/src/interface.cairo	`fd6a3ff332033de3334501a4edb464f6c0d95322`
contracts/account/src/lib.cairo	`ff8597b66699df522c63b47749bf7fcbfd458b2d`
contracts/lib/src/account.cairo	`1158135c3a7b110d7e66e98d44fb61181ce6a81c`
contracts/lib/src/array_ext.cairo	`d004d0dbd3137b7eb927597428e034d4b0eb33dd`
contracts/lib/src/asserts.cairo	`d708eb44f1a8f3893c9ec44123efd57e41f1bde5`
contracts/lib/src/calls.cairo	`d9a7eb8383be360515f12c061183944b3c75866d`
contracts/lib/src/erc165.cairo	`695ffb6300c7946bd8767aeb6da614999d3e09c4`
contracts/lib/src/lib.cairo	`3b8608f652132102fb08ce4161f2a76e4a66eeb7`
contracts/lib/src/outside_execution.cairo	`0381d981e465c3e99fd6b541503ed147337d5ade`
contracts/lib/src/upgrade.cairo	`7846cbd9b42f4ef39931d92fa1c14e493078b663`
contracts/lib/src/version.cairo	`673665ed1a78a5f7f2508bf0b5dfbc1c739082b6`
contracts/multisig/src/argent_multisig.cairo	`a2c2a0562e194659c65382fb06e3f5e45a1604b1`
contracts/multisig/src/interface.cairo	`ecf712fcba7f677475fbab2ed38d9d3ba5d93f84`
contracts/multisig/src/lib.cairo	`f839aa19ce20a8b3710bac53d8a8fb1070033e03`
contracts/multisig/src/signer_signature.cairo	`ef0eec9201757a1afabbc73aa2c67fbcbb58489a`

Note: The `contracts/multisig/src/argent_multisig_storage.cairo` file with the original SHA-1 hash `2702b447f5257a126b7bee3188f5fbf9bfc960d0`, `contracts/lib/src/erc1271.cairo` file with the original SHA-1 hash `49dc11614809a07dc8362c0b8f927ff7da9454e2`, and

`contracts/lib/src/span_serde.cairo` file with the original SHA-1 hash `a52d8526c5f3d5016d3893f88545e59728f12302` were removed from the final version of the audit repository.

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself 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 specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.