Argent Account & Argent Multisig Starknet Transaction V3 Updates

Date	January 2024
Auditors	Heiko Fisch, George Kobakhidze

1 Executive Summary

This report presents the results of our engagement with Argent to review updates in **Argent Account** and **Argent Multisig** for Starknet Transaction v3.

The review was conducted from **January 15–23, 2024**, by **George Kobakhidze** and **Heiko Fisch**. A total of 14 person-days were spent. Due to Argent's time constraints regarding the release, potential findings were reported informally during the first week of the engagement, with the client implementing fixes and improvements as discussed. At the end of the week, we reviewed these changes and finished the assessment. The report was compiled in the first half of the following week.

Argent is a provider of wallets in the Ethereum ecosystem, including its L2s. This audit focused on their smart contract wallet offering for Starknet, Argent Account (v0.3.1), and their multisig, Argent Multisig (v0.1.1), as well as the auxiliary libraries used for these. Starknet recently introduced the new transaction version 3, which Argent wanted to support. We had already reviewed an earlier version of the Argent Account and Multisig for Starknet, and it was our pleasure to work with Argent again to review the changes that were made to support the new Starknet transactions, along with a few smaller improvements that had become possible due to new releases of the Cairo language.

As last time, we found the code exceptionally well-written, -documented, and extensively tested. All our findings, as detailed in Section 4, have been fixed in the final version we considered in this audit with the commit hash `3b5e45cf28113f7389a716ef9faabfbd79ce45f9`.

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

We reviewed the client's repository at the commit hash `cb01c6a3c8f343ef195ebb7094361b674fe43a37`. All findings that we identified have been fixed in the version with commit hash `3b5e45cf28113f7389a716ef9faabfbd79ce45f9`. Only Cairo files – and most of them – have been in scope for this audit; a detailed list, together with their SHA-1 hashes in the initial and final version considered in this engagement, can be found in the Appendix. Naturally, we focused on the changes that have been introduced since our last review.

3 System Overview

Since the primary focus of this review was the changes introduced with new functionality available in Starknet and Cairo, they will be discussed first.

3.1 Transaction V3

The main anticipated change with this update is the support for v3 transactions. As can be found in Starknet documentation and the relevant SNIP-8, this new transaction type will allow users to utilize `STRK` as the gas token for transaction fees. As a result, some changes had to be made to Argent smart contracts, specifically in the functions validating invoking, deploying, and declaring transactions.

Additionally, since Argent accounts have Guardian addresses that can spend a limited amount of funds for transaction fees without approval of the owner during escapes, new logic had to be introduced to limit that for v3 transactions as well. Namely, v3 transactions introduce concepts such as `max_price_per_unit` and `max_amount` for L1 and L2 gas separately, as well as a `tip` for L2 gas, whereas v1 transactions just have a singular value `max_fee`. Therefore, new limits are needed to restrict the amount of gas a malicious Guardian can spend with a v3 transaction.

3.2 New Cairo Starknet Contract Functionalities

Along with new transaction types, Cairo Starknet contracts can now utilize new features that allow for a better developer experience. These include:

- Usage of `selector!`. This allows for easier access to a `starknet_keccak` of a string where needed, such as when computing selectors of functions and hashes for signature data.
- Usage of events. With this new version, events can now be emitted from a contract with a simple `self.emit`, as opposed to calling them like a function as in previous versions.
- Usage of `self`. Now smart contracts may utilize a special keyword called `self` that allows for easier access for a contract to interact with its internal functions, state variables, events, and more.

3.3 General System Overview

The rest of the system functionality outside of the changes remains the same as in the last version; a description may be found in our previous audit report here.

4 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.

4.1 Lack of Fee Limits for V3 Transactions Major ✓ Fixed

Resolution

Fixed in PR-266 by introducing limits on the v3 transaction fees through two separate maximums:

- `MAX_ESCAPE_TIP_STRK = 1 STRK` which limits the result of multiplying `tip * L2_GAS.max_amount`.
- `MAX_ESCAPE_MAX_FEE_STRK = 50 STRK` which limits the sum of all of the following:
  - `L1_GAS.max_price_per_unit * L1_GAS.max_amount`,
  - `L2_GAS.max_price_per_unit * L2_GAS.max_amount`,
  - `tip * L2_GAS.max_amount`, the tip described above.

Description

Starknet transactions have fields to specify the maximum amount of fees the sequencer may take. For v1 transactions, this is just one field with the name `max_fee` and unit WEI (i.e., 10^{-18} ETH). For the newly introduced v3 transactions, the situation is a bit more complicated. First of all, there are two types of fees: `L1_GAS` and `L2_GAS`. The former is needed to cover the gas costs on L1 that the transaction produces, the second is supposed to cover the L2 costs and will be utilized in the upcoming fee market. For each of these two fee types, a transaction specifies the `max_amount` and the `max_price_per_unit`. In addition to that, there is also a `tip` field to help facilitate the market. It is noteworthy that the `max_price_per_unit` fields – even for `L1_GAS` – and the `tip` will be specified in 10^{-18} STRK/gas. Another point worth highlighting is that, as Starknet has built-in account abstraction, the fee for a transaction is paid by the account.

Currently, the only sequencer is operated by StarkWare and charges a fair price for the L1 costs. (And L2 fees are not being collected yet.) Hence, even if a transaction specifies a very high fee limit, the sequencer takes only what is really needed and not everything that the transaction limit(s) would allow. For the sake of brevity, let us call such a sequencer "nice". In a more decentralized Starknet future, there will probably be more sequencers, and they may not necessarily be nice, meaning they may take more in fees than what the L1 costs demand of them. Also, it is not impossible that the rules for the StarkWare sequencer change at some point in the future (although it seems reasonable to assume that this would be properly announced). But – to summarize this discussion – currently there is only sequencer, and it is nice. The attack we describe below requires a "non-nice" sequencer – and, as we will explain shortly, a malicious Guardian – and is therefore, at the time of writing this report, not feasible, even assuming the Guardian acts with malice.

As explained in more detail in the System Overview of our previous report, there is an escape mechanism which (1) allows users to reclaim control of their account if the Guardian fails to cooperate and (2) allows Guardians to assign a new owner if the original owner lost access to their key. Crucially and unlike other account activities, escape-related actions require only a single signer. Hence, a general attack scenario that the account should implement protective measures against is a malicious Guardian trying to drain the account by signing an escape transaction with an excessive fee limit. A similar situation arises if, instead of the Guardian being malicious, a third party comes into possession of the owner's private key, but to keep the discussion more concise, we'll consider this subsumed under "malicious Guardian." As mentioned above, such an attack is not possible with a nice sequencer, but – ideally – the account contract should not rely on that.

Examining the relevant code, we see that the fee restriction logic for v1 transactions – which is known from earlier versions of the contract – is still present:

src/account/argent_account.cairo:L47-L48

```
/// Limits fee in escapes
const MAX_ESCAPE_MAX_FEE: u128 = 50000000000000000; // 0.05 ETH
```

src/account/argent_account.cairo:L772-L774

```
} else if tx_info.version == TX_V1 || tx_info.version == TX_V1_ESTIMATE {
    // other fields not available on V1
    assert(tx_info.max_fee <= MAX_ESCAPE_MAX_FEE, 'argent/max-fee-too-high');
```

And there is a limit on the total tip, i.e., `tip * L2_GAS.max_amount`, for v3 transactions:

src/account/argent_account.cairo:L49-L50

```
/// Limits tip in escapes
const MAX_ESCAPE_TIP: u128 = 1_000000000000000000; // 1 STRK
```

src/account/argent_account.cairo:L758-L771

```
// Limit the maximum tip while escaping (max_fee returns 0 on TX_V3)
let max_l2_gas: u64 = loop {
    match tx_info.resource_bounds.pop_front() {
        Option::Some(r) => { if *r.resource == 'L2_GAS' {
            break *r.max_amount;
        } },
        Option::None => {
            // L2_GAS not found
            break 0;
        }
    };
};
let max_tip = tx_info.tip * max_l2_gas.into();
assert(max_tip <= MAX_ESCAPE_TIP, 'argent/tip-too-high');
```

However, no limit is imposed on the amount of STRK for `L1_GAS` or `L2_GAS`. Regarding `L1_GAS`, this means that a malicious Guardian could specify an excessive `max_price_per_unit` in an escape transaction and – with the help of a non-nice sequencer – drain the account's entire STRK balance. Since the sequencer can pocket the difference between `max_price_per_unit` and what is really needed on L1, it is also conceivable that the two parties collude for an attack.

For `L2_GAS`, the situation is more difficult to assess because the fee market has not been implemented yet. It is very well possible that the "base fee" will be set by the network – similar to the base fee on Ethereum, see EIP-1559. In this case, constraining only the tip, as is currently the case, would be sufficient to prevent the `L2_GAS` part of this kind of attack, as long as one is comfortable with risking to pay any fee the market demands. Nevertheless, as the details of the L2 fee mechanism have not yet been specified, we advise caution.

Recommendation

For both `L1_GAS` and `L2_GAS`, the amount of STRK that can be spent on fees should be limited, similar to the limit on `max_fee` for v1 transactions. There are several different – and equally viable – ways to do this: separately for `L1_GAS` and `L2_GAS` or together; including the tip in a (higher) limit or handling it separately.

One aspect of the L2 fees that has, to the best of our knowledge, not been decided yet is whether the `tip` will be considered part of the `max_price_per_unit` (similar to EIP-1559) or if it can be on top of that. Hence, to be on the safe side, the latter should be considered possible, and the tip should not be left unconstrained.

4.2 Newer Cairo Version Available Minor ✓ Fixed

Resolution

When we informed the client about this, we learned that they had already updated the Cairo version on the development branch to 2.4.3 (commit `72c14f0`), so they were aware of this.

Description

The Cairo version used in the commit hash specified for the audit is 2.4.0. Cairo receives updates in quick succession, and, at the time of conducting the review, newer versions are available. Version 2.4.1, in particular, comes with some bug fixes.

Recommendation

We recommend utilizing the latest available Cairo version or at least a version without known bugs.

4.3 Discrepancy Between Actual `OUTSIDE_EXECUTION_TYPE_HASH` and Comments Minor ✓ Fixed

Resolution

Fixed in PR-266 by adjusting the comments, removing the `OutsideCall` struct, and using `selector!()` to directly calculate the `OUTSIDE_EXECUTION_TYPE_HASH` instead of using a hardcoded constant.

Description

In the `outside_execution.cairo` file, we have a hardcoded value `const OUTSIDE_EXECUTION_TYPE_HASH: felt252 = 0x11ff76fe3f640fa6f3d60bbd94a3b9d47141a2c96f87fdcfbeb2af1d03f7050`. The comment above it indicates that to derive it, we need to use the hash `H('OutsideExecution(caller:felt,nonce:felt,execute_after:felt,execute_before:felt,calls_len:felt,calls:Call*)')`, where "H" would be the `starknet_keccak` (which is confirmed with other values):

src/common/outside_execution.cairo:L33-L34

```
// H('OutsideExecution(caller:felt,nonce:felt,execute_after:felt,execute_before:felt,calls_len:felt,calls:Call*)')
const OUTSIDE_EXECUTION_TYPE_HASH: felt252 = 0x11ff76fe3f640fa6f3d60bbd94a3b9d47141a2c96f87fdcfbeb2af1d03f7050;
```

However, doing the above hash results in the value `0x28df6ab27eb241200f2ba2177e1ad2c81bd92b71bfd8b8fa40ced4d3b55d66d`. If we add the `Call` struct to the string as well, we get

`'OutsideExecution(caller:felt,nonce:felt,execute_after:felt,execute_before:felt,calls_len:felt,calls:Call*)Call(to:felt,selector:felt,calldata_len:felt,calldata:felt*)'`
, which when hashed yields `0x2838a3633cc7cd97bbf5b9f800d77b6d891154b8abdf6f41132c40e5a9ace2c` that also doesn't match.

Finally, if we observe the function `hash_outside_execution`, we can see that it computes the selector with the string `'OutsideExecution(caller:felt,nonce:felt,execute_after:felt,execute_before:felt,calls_len:felt,calls:OutsideCall*)OutsideCall(to:felt,selector:felt,calldata_len:felt,calldata:felt*)'`
:

src/common/outside_execution.cairo:L101-L103

```
selector!(
    "OutsideExecution(caller:felt,nonce:felt,execute_after:felt,execute_before:felt,calls_len:felt,calls:OutsideCall*)OutsideCall(to
)
```

If we take this string and hash it, we indeed get `0x11ff76fe3f640fa6f3d60bbd94a3b9d47141a2c96f87fdcfbeb2af1d03f7050`. However, this does ask for `OutsideCall` in the `OutsideExecution` struct, whereas the code may suggest it should be just `Call`.

Recommendation

While there is no real impact as the hash constant is the correct one, it would be beneficial to get the comments and the code in sync with the intended values.

4.4 Self-Written Versions of `get_execution_info` and `get_tx_info` ✓ Fixed

Resolution

Fixed in PR-266 by using the `get_execution_info` and `get_tx_info` functions from the `starknet` package.

Description

Functions that retrieve the execution and transaction information have been implemented in `transaction_version.cairo`:

src/common/transaction_version.cairo:L37-L45

```
#[inline(always)]
fn get_execution_info() -> Box<starknet::info::v2::ExecutionInfo> {
    starknet::syscalls::get_execution_info_v2_syscall().unwrap_syscall()
}

#[inline(always)]
fn get_tx_info() -> Box<starknet::info::v2::TxInfo> {
    get_execution_info().unbox().tx_info
}
```

However, the same functionality is provided by the `starknet` package, which is used anyway.

Recommendation

Although there is no difference in behavior, we recommend utilizing the functions from Cairo's starknet package. The self-written versions can then be removed from `transaction_version.cairo`.

4.5 `__validate_deploy__` Function Doesn't Have Its Own Transaction Version Check ✓ Fixed

Resolution

Fixed in PR-266 by decoupling the transaction checks through the introduction of `assert_correct_deploy_account_version()` check.

Description

In the current version of Starknet, accounts have required functions that validate specific types of transactions, such as `INVOKE`, `DECLARE`, and `DEPLOY_ACCOUNT`, as seen here.

In `argent_account.cairo` there are indeed those functions, where it is also checked that the transaction going through those functions is compliant with specific transaction parameters:

src/account/argent_account.cairo:L202-L205

```
fn __validate__(ref self: ContractState, calls: Array<Call>) -> felt252 {
    assert_caller_is_null();
    let tx_info = get_tx_info().unbox();
    assert_correct_invoke_version(tx_info.version);
```

src/account/argent_account.cairo:L329-L331

```
fn __validate_declare__(self: @ContractState, class_hash: felt252) -> felt252 {
    let tx_info = get_tx_info().unbox();
    assert_correct_declare_version(tx_info.version);
```

src/account/argent_account.cairo:L337-L341

```
fn __validate_deploy__(
    self: @ContractState, class_hash: felt252, contract_address_salt: felt252, owner: felt252, guardian: felt252
) -> felt252 {
    let tx_info = get_tx_info().unbox();
    assert_correct_invoke_version(tx_info.version);
```

The checks:

src/common/transaction_version.cairo:L14-L28

```
#[inline(always)]
fn assert_correct_invoke_version(tx_version: felt252) {
    assert(
        tx_version == TX_V3 || tx_version == TX_V1 || tx_version == TX_V3_ESTIMATE || tx_version == TX_V1_ESTIMATE,
        'argent/invalid-tx-version'
    )
}

#[inline(always)]
fn assert_correct_declare_version(tx_version: felt252) {
    assert(
        tx_version == TX_V3 || tx_version == TX_V2 || tx_version == TX_V3_ESTIMATE || tx_version == TX_V2_ESTIMATE,
        'argent/invalid-declare-version'
    )
}
```

However, while `__validate__` and `__validate_declare__` have their own transaction check, namely `assert_correct_invoke_version` and `assert_correct_declare_version`, `__validate_deploy__` reuses the same check as the `INVOKE` transaction – `assert_correct_invoke_version`.

Recommendation

While that is technically fine (although the documentation isn't consistent in all places about this) as it indeed does support v3 and v1 transactions, it would be beneficial to have a separate check just for the `DEPLOY_ACCOUNT` transactions for better maintainability of the codebase.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash in initial version (`cb01c6a`)	SHA-1 hash in final version (`3b5e45c`)
src/account/argent_account.cairo	`6a6ff6fcea1c8886bb7fea8bd5c86b9636dbbe27`	`d04bd21dbe98b492c8942e52c4a03270288599d9`
src/account/escape.cairo	`712bbc5ab869e52a0dd8a8fb12648b1c205b219a`	`712bbc5ab869e52a0dd8a8fb12648b1c205b219a`
src/account/interface.cairo	`c99207364b2db53c102039e082ce6ea6b7680d71`	`c99207364b2db53c102039e082ce6ea6b7680d71`
src/common/account.cairo	`b789ff06b8c72637bcd3a7564ab2c98a348f2a9d`	`b789ff06b8c72637bcd3a7564ab2c98a348f2a9d`
src/common/array_ext.cairo	`f89e6a0a3c2eb4f10f38581b27ee533206787499`	`f89e6a0a3c2eb4f10f38581b27ee533206787499`
src/common/asserts.cairo	`a14088bab4704ac039234391c51f32fb3b22ceee`	`a14088bab4704ac039234391c51f32fb3b22ceee`
src/common/calls.cairo	`9e87de8b777550a42597e670f09a1928c0fff663`	`9e87de8b777550a42597e670f09a1928c0fff663`
src/common/erc165.cairo	`a16e4692ea9367b5b1ad768fd23518c144d227bd`	`a16e4692ea9367b5b1ad768fd23518c144d227bd`
src/common/outside_execution.cairo	`8f2bca22d117b94ac53e5519b9615d320caa0965`	`666b40d3dd035d5e1df6515eb994ace9a5a822e6`
src/common/transaction_version.cairo	`104db449a6b4705d31f2ee94bef708cae14c56f0`	`8b0d320ed245d7b176d7e08b702f20d63e8605b1`
src/common/upgrade.cairo	`575f17add2ef5d9a1dd21bfc3d129bf15ecd5216`	`575f17add2ef5d9a1dd21bfc3d129bf15ecd5216`
src/common/version.cairo	`673665ed1a78a5f7f2508bf0b5dfbc1c739082b6`	`673665ed1a78a5f7f2508bf0b5dfbc1c739082b6`
src/multisig/argent_multisig.cairo	`9d0e563882c1668fea1e021f3e18d41f7a3e5740`	`a67b25a848f469aa9b76f4aa9fddb0daacc2fb13`
src/multisig/interface.cairo	`7ee3f16ed71bc264e3cc84a7baf05854ae049ded`	`7ee3f16ed71bc264e3cc84a7baf05854ae049ded`
src/multisig/signer_signature.cairo	`e30651ba51289e1aeaed0ed8a6608a874ba7979d`	`e30651ba51289e1aeaed0ed8a6608a874ba7979d`

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 bug-free 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.

A.2.1 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.

A.2.2 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.

A.2.3 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.