Security Review Report
NM-0064-StarkGate (Layer 1 and Layer 2)

NETHERMIND

(Nov 10, 2022)

Contents

1 Executive Summary

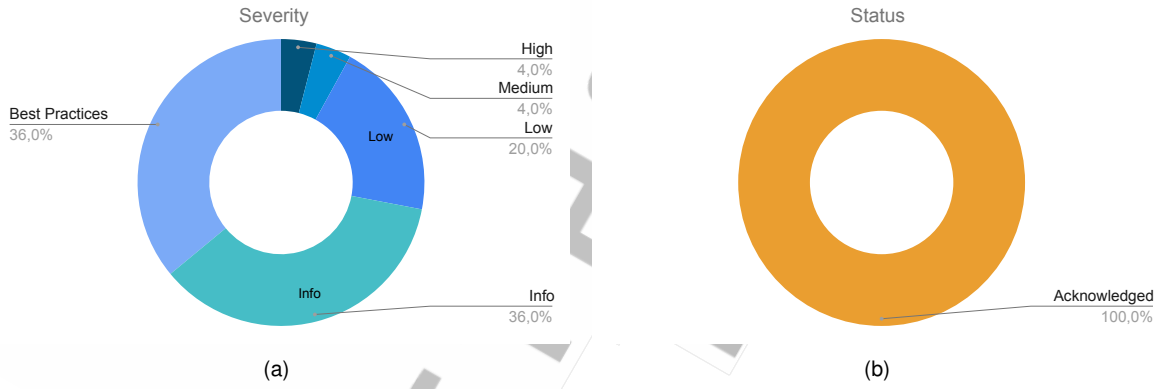
This document presents the security review of StarkWare's StarkGate token bridge implementation performed by Nethermind. StarkGate is an Ether and ERC20 bridge allowing assets to be transferred between Ethereum and StarkNet. The codebase comprises 989 lines of Solidity and 1151 lines of Cairo. The audit was completed primarily with manual analysis of the codebase, as well as using automated analysis tools. The purpose of Starkgate is to facilitate user's ability to conduct their transactions with their ETH and ERC-20 tokens that reside on L1, via the StarkNet Alpha network and its STARK-based computational compression capabilities. At the moment of writing, four different assets are supported for the bridge (ETH, WBTC, USDC and USDT) with a TVL bigger than 650,000 USD. The bridge can be currently used from StarkGate Testnet and StarkGate Mainnet. The bridge makes use of proxies and governance to allow for management of the bridge parameters as well as authorized implementation upgrades. This codebase can be broken into three separate components: (1) bridge logic; (2) governance logic; and, (3) proxy logic.

The bridge logic (1) is well written and has minimal complexity which is ideal for security. Appropriate checks and input validations are done to ensure that only correct data can be sent between layers, and while the code is still under development a restriction on the max deposit and max TVL per asset is a good design decision.

The governance logic (2) is functional, but raises some concerns. Particularly, the functional inconsistencies between the L1 and L2 implementation, as well as a lack of voting which makes this governance implementation closer to a multiple owner setup which will need only one malicious governor to cause damage to the bridge. More details on governance concerns can be found in Section 7 in this report.

The proxy logic (3) is also functional, but has a high level of complexity which appears to be unnecessary for the relatively simple asset transfer logic. This proxy introduces the concept of EICs (External Initializer Contracts), which allow a dedicated contract to hold the initialization logic that would normally be located in the implementation. The behavior of upgrading an implementation can differ depending on if an EIC is used, and for each approach there will be another different behavior depending on if the contract is already initialized or not. Unnecessary complexity should be avoided where possible, especially where delegated/library calls are used as it could cause significant damage if something unexpected were to occur. Further details on proxy concerns can also be found in Section 7.

Since September 2022, approximately 50% of all exploited funds have come from bridge attacks. The large concentration of assets on bridge contracts makes them a significant target for malicious parties. The Nethermind team strongly recommends **the StarkGate bridge contract to be formally verified to ensure that all the properties and invariants hold**. The Nethermind team also recommends the adoption of **real-time monitoring of the bridge** smart contracts to be able to quickly identify and respond in the case of attacks. **During the initial audit, we point out 25 potential issues** summarized in the figure below.



Distribution of issues: Critical (0), **High** (1), **Medium** (1), **Low** (5), **Undetermined** (0), **Informational** (9), **Best Practices** (9).
Distribution of status: Unresolved (25)

Summary of the Audit

Audit Type	Security Review
Initial Report	Nov. 10, 2022
Response from Client	-
Final Report	-
Methods	Manual Review, Automated Analysis
Repository	StarkGate Contracts
Commit Hash (Initial Audit)	fd5ad0bd1c027040a014081943aefbe8664f71c6
Documentation	README.md
Documentation Assessment	Low
Test Suite Assessment	Low

In summary, the bridge logic that handles asset transfer between chains shows a robust design. However, the design choices and complexity surrounding the governance logic and proxy logic introduce risk to the protocol that can be avoided with a more simple design approach. **On the other hand, the test suite is limited and required effort from our team in order to be executed**.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the protocol. Section 4 presents the summary of issues. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses some concerns the audit team has. Section 8 discusses the documentation provided for this audit. Section 9 discusses the test suite. Section 10 discusses issues report by static analysis tools. Section 11 concludes the document.

2 Audited Files

Layer-1 Solidity Contracts

	Contract	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	src/starkware/cairo/eth/CairoConstants.sol	5	1	20.0%	1	7
2	src/starkware/starknet/apps/starkgate/eth/StarknetBridgeConstants.sol	9	2	22.2%	1	12
3	src/starkware/starknet/apps/starkgate/eth/StarknetTokenBridge.sol	173	44	25.4%	30	247
4	src/starkware/starknet/apps/starkgate/eth/StarknetTokenStorage.sol	51	4	7.8%	15	70
5	src/starkware/starknet/apps/starkgate/eth/StarknetEthBridge.sol	24	7	29.2%	7	38
6	src/starkware/starknet/apps/starkgate/eth/Transfers.sol	46	14	30.4%	8	68
7	src/starkware/starknet/apps/starkgate/eth/StarknetERC20Bridge.sol	20	4	20.0%	4	28
8	src/starkware/starknet/solidity/IStarknetMessagingEvents.sol	38	7	18.4%	6	51
9	src/starkware/starknet/solidity/IStarknetMessaging.sol	24	25	104.2%	5	54
10	src/starkware/solidity/upgrade/ProxyGovernance.sol	24	35	145.8%	8	67
11	src/starkware/solidity/upgrade/ProxyStorage.sol	7	12	171.4%	4	23
12	src/starkware/solidity/upgrade/StorageSlots.sol	11	22	200.0%	4	37
13	src/starkware/solidity/upgrade/Proxy.sol	150	122	81.3%	40	312
14	src/starkware/solidity/libraries/NamedStorage.sol	87	6	6.9%	12	105
15	src/starkware/solidity/libraries/Addresses.sol	22	19	86.4%	4	45
16	src/starkware/solidity/tokens/ERC20/ERC20.sol	107	1	0.9%	21	129
17	src/starkware/solidity/tokens/ERC20/IERC20.sol	15	5	33.3%	8	28
18	src/starkware/solidity/tokens/ERC20/IERC20Metadata.sol	7	13	185.7%	4	24
19	src/starkware/solidity/components/GenericGovernance.sol	29	4	13.8%	9	42
20	src/starkware/solidity/components/GovernanceStorage.sol	5	5	100.0%	1	11
21	src/starkware/solidity/components/Governance.sol	58	41	70.7%	15	114
22	src/starkware/solidity/interfaces/ExternalInitializer.sol	5	1	20.0%	2	8
23	src/starkware/solidity/interfaces/MGovernance.sol	8	4	50.0%	2	14
24	src/starkware/solidity/interfaces/ContractInitializer.sol	8	25	312.5%	5	38
25	src/starkware/solidity/interfaces/ProxySupport.sol	41	26	63.4%	11	78
26	src/starkware/solidity/interfaces/BlockDirectCall.sol	11	7	63.6%	3	21
27	src/starkware/solidity/interfaces/Identity.sol	4	4	100.0%	1	9
	Total	**989**	**460**	**46.5%**	**231**	**1680**

Layer-2 Cairo Contracts

	Contract	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	src/starkware/starknet/apps/starkgate/cairo/token_bridge_interface.cairo	21	0	0.0%	9	30
2	src/starkware/starknet/apps/starkgate/cairo/token_bridge.cairo	214	33	15.4%	45	292
3	src/starkware/starknet/std_contracts/ERC20/permitted.cairo	28	3	10.7%	8	39
4	src/starkware/starknet/std_contracts/ERC20/IERC20.cairo	23	0	0.0%	10	33
5	src/starkware/starknet/std_contracts/ERC20/mintable_token_interface.cairo	9	0	0.0%	3	12
6	src/starkware/starknet/std_contracts/ERC20/ERC20_base.cairo	143	13	9.1%	36	192
7	src/starkware/starknet/std_contracts/ERC20/ERC20.cairo	139	12	8.6%	32	183
8	src/starkware/starknet/std_contracts/upgradability_proxy/initializable.cairo	23	0	0.0%	5	28
9	src/starkware/starknet/std_contracts/upgradability_proxy/governance.cairo	117	12	10.3%	26	155
10	src/starkware/starknet/std_contracts/upgradability_proxy/impl_contract_b.cairo	38	2	5.3%	6	46
11	src/starkware/starknet/std_contracts/upgradability_proxy/proxy_impl.cairo	232	32	13.8%	29	293
12	src/starkware/starknet/std_contracts/upgradability_proxy/proxy.cairo	58	0	0.0%	6	64
13	src/starkware/starknet/std_contracts/upgradability_proxy/Initializable_interface.cairo	13	0	0.0%	2	15
14	src/starkware/starknet/std_contracts/upgradability_proxy/test_eic.cairo	32	0	0.0%	4	36
15	src/starkware/starknet/std_contracts/upgradability_proxy/finalizable.cairo	23	0	0.0%	5	28
16	src/starkware/starknet/std_contracts/upgradability_proxy/impl_contract_a.cairo	38	2	5.3%	6	46
	Total	**1151**	**109**	**9.5%**	**232**	**1492**

3 Protocol Summary

This section presents a high-level description of the protocol. Users deposit their funds on Layer-1, where they are locked in escrow. An equivalent amount of tokens representing the deposited assets are minted on Layer-2 to the user's nominated address. When the user decides to withdraw their funds back to Layer-1, their Layer-2 tokens are burned, and an equivalent amount of Layer-1 tokens are returned to the user. This allows for a 1:1 ratio between the Layer-1 and Layer-2 tokens. The flow diagram presented in Fig. 1 was made by the Nethermind team during the audit, and was created by manually reviewing the code and following contract calls for all user actions.

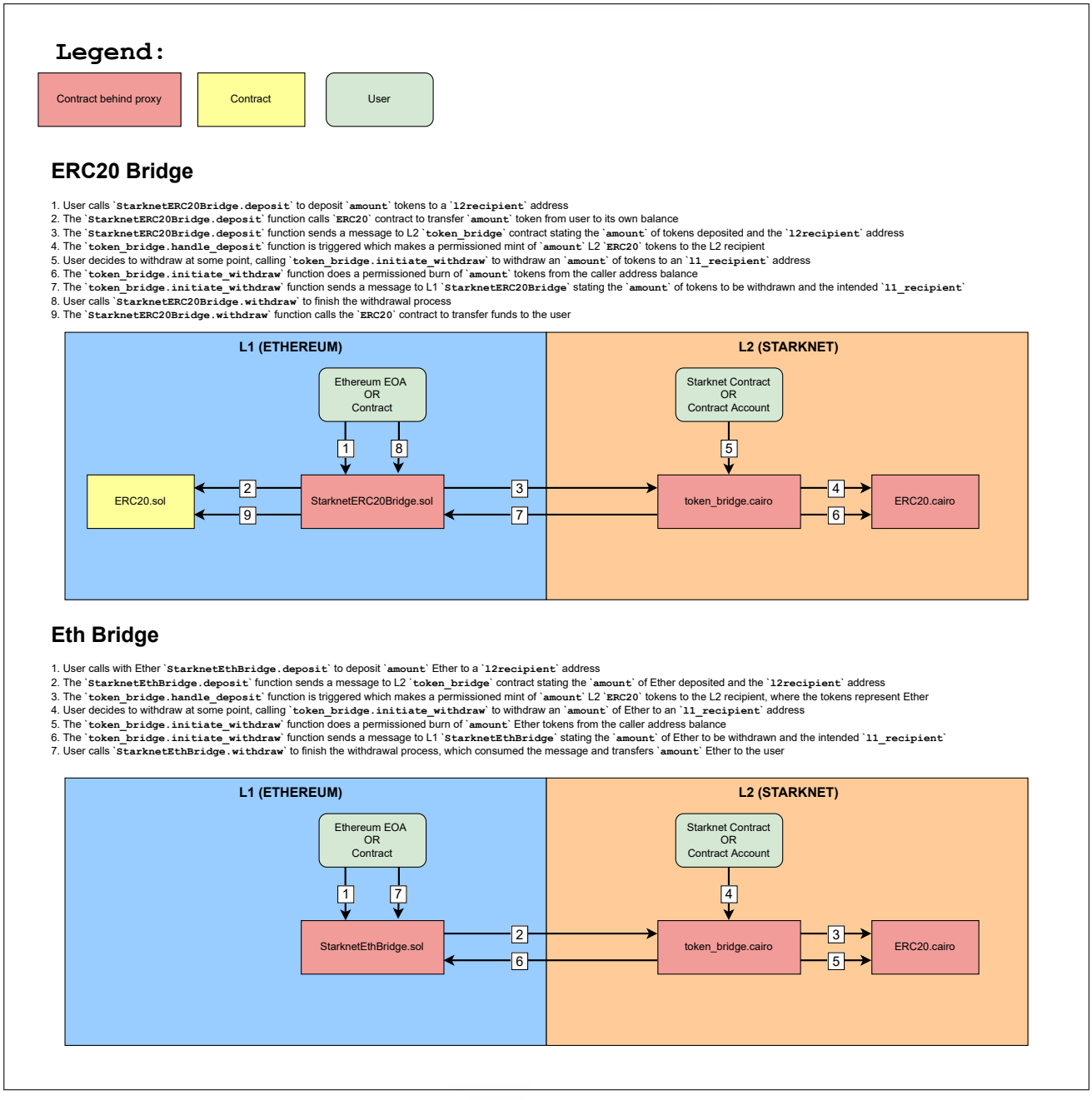


Figure 1: StarkGate's Flow Diagram constructed by the Nethermind team.

4 Summary of Issues

	Finding	Severity	Update
1	Fees are not returned after canceling a message	High	Unresolved
2	Proxy may not accept message cancellation refunds	Medium	Unresolved
3	Incorrect value for ERC20 constant `MAX_DECIMALS`	Low	Unresolved
4	No commitment to code executed in external initializer	Low	Unresolved
5	Storage slot calculated from spelling error	Low	Unresolved
6	Unable to upgrade contract without revealing new implementation	Low	Unresolved
7	Wrong use `assert_le(...)`	Low	Unresolved
8	Added implementation time delay value may be overwritten	Info	Unresolved
9	Function `setAddressValueOnce()` can overwrite values	Info	Unresolved
10	Gas saving opportunity in `NamedStorage` library	Info	Unresolved
11	Governance Model invariants are not checked	Info	Unresolved
12	Governance can change `l2TokenBridge` without restrictions	Info	Unresolved
13	Governor removal is missing a proposal process	Info	Unresolved
14	Prevent deposits and withdrawals of zero value	Info	Unresolved
15	Storage variable `decimals` can be `immutable`	Info	Unresolved
16	Unused function import	Info	Unresolved
17	Constant is never used	Best Practices	Unresolved
18	External proxy governance initialization	Best Practices	Unresolved
19	Function `implementationIsFrozen(...)` returns `false` when no implementation is set	Best Practices	Unresolved
20	Gas Optimization in `_acceptGovernance(...)`	Best Practices	Unresolved
21	No check for caller address	Best Practices	Unresolved
22	Old Solidity version	Best Practices	Unresolved
23	Returning number instead of a boolean	Best Practices	Unresolved
24	Unchecked `final` boolean argument in `upgrade_to(...)`	Best Practices	Unresolved
25	Unlocked Pragma	Best Practices	Unresolved

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 is a measure of 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 the finding were to be exploited by an attacker. 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, 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 formally pass to the client;

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 Cross-Layer Issues

6.1.1 [Info] Added implementation time delay value may be overwritten

File(s): Proxy.sol, proxy_impl.cairo

Description: The functions `addImplementation(...)` (L1) and `add_implementation(...)` (L2) are used to add a new implementation. The hash of the implementation code, initialization data and the `finalize` flag are used as a key to locate the upgrade time delay for a given implementation. It is possible to overwrite the time delay of an already-added implementation by adding it again, which will effectively reset the upgrade timer. This can lead to a scenario where a governor can postpone an implementation upgrade by adding the same implementation again just before the time delay is about to complete.

Recommendation(s): Consider checking if the implementation has already been added to avoid overwriting the time delay value.

Status: Unresolved

Update from the client:

6.1.2 [Info] Governor removal is missing a proposal process

File(s): `src/starkware/*`

Description: When adding a governor, a candidate must be proposed and then another governor will approve the candidate. This process prevents one governor from adding potentially unwanted governors to the protocol. Such a process does not exist when removing governors, allowing one governor to remove all others without the need for approval. If one governor address is compromised then all other governors may be put at risk. This applies to both L1 and L2 governance implementations.

Recommendation(s): Consider adding a propose-approve process to the removal of governors.

Status: Unresolved

Update from the client:

6.1.3 [Info] Prevent deposits and withdrawals of zero value

File(s): `src/starkware/*`

Description: For both the Ether and ERC20 bridges it is possible to deposit and withdraw amounts of zero. This makes no change to the user's asset balance but costs the user fees regardless, and will also emit unnecessary events on both L1 and L2.

Recommendation(s): Consider adding checks to the L1 deposit functions (Ether and ERC20) and L2 withdraw function to prevent amounts of zero from being transferred.

Status: Unresolved

Comment from the client:

6.1.4 [Best Practices] External proxy governance initialization

File(s): src/starkware/

Description: The proxy implementation on `L2` must initialize its governance through the external function `init_governance(...)` rather than through the constructor. This may add unnecessary complexity to the proxy contract deployment. Additionally, on `L1` the proxy governance is initialized through the constructor. Inconsistencies between initialization of proxies across layers may lead to confusion and unintended misconfiguration at deployment time.

Recommendation(s): Consider unifying implementation of governance initialization across the layers by moving `init_governance(...)` function into the Proxy's constructor on `L2`.

Status: Unresolved

Comment from the client:

6.2 L1 Issues

6.2.1 [Low] Storage slot calculated from spelling error

File(s): StarknetMessaging.sol, StorageSlots.sol

Description: The storage slots used by the bridge are determined by strings that are passed through `keccak256(...)`. The strings that are used contain spelling errors. If the spelling error is corrected and then upgraded over an existing implementation that originally had the error then access to that storage location will be lost. Carefully consider the impact of correcting these errors. A list of incorrect strings used to calculate storage slots is shown below.

```
// src/starkware/starknet/solidity/StarknetMessaging.sol#L16-L17
string constant L1L2_MESSAGE_MAP_TAG = "STARKNET_1.0_MSGING_L1TOL2_MAPPPING_V2";
string constant L2L1_MESSAGE_MAP_TAG = "STARKNET_1.0_MSGING_L2TOL1_MAPPPING";
```

```
// src/starkware/starknet/solidity/upgrade/StorageSlots.sol#L11
// The address of the slot is keccak256("'StarkWare2020.CallProxy.Implemntation.Slot'").
```

Recommendation(s): The team must discuss the best strategy to solve this issue. It can lead to potential risks in future upgrades.

Status: Unresolved

Comment from the client:

6.2.2 [Best Practices] Old Solidity version

File(s): src/starkware/

Description: All Solidity files have the version pragma of `^0.6.12`, where the latest release on the `0.6` version branch was on July 22nd, 2020. It is recommended to use a recent Solidity version to avoid known bugs from previous versions and utilize new features added with the later versions.

Recommendation(s): Consider changing the Solidity version to `0.8.*`.

Status: Unresolved

Comment from the client:

6.2.3 [Best Practices] Unlocked Pragma

File(s): src/starkware/

Description: All Solidity contracts have an open version pragma to allow versions from `0.6.12` to the latest `0.6.*` version.

Recommendation(s): For consistency between testing, deployments and implementation upgrades it is recommended to lock the contracts to a specific Solidity version.

Status: Unresolved

Comment from the client:

6.3 src/starkware/solidity/tokens/ERC20/ERC20.sol

6.3.1 [Info] Storage variable `decimals` can be `immutable`

File(s): ERC20.sol

Description: The storage variable `decimals` is only set once from the constructor during contract deployment and cannot be changed afterwards. Declaring this variable as immutable will save gas during deployment and will remove the need for an unnecessary `SLOAD` when calling `decimals(...)`.

Recommendation(s): Consider declaring the `decimals` state variable as `immutable`.

Status: Unresolved

Comment from the client:

6.4 src/starkware/solidity/upgrade/StorageSlots.sol

6.4.1 [Best Practices] Constant is never used

File(s): StorageSlots.sol

Description: The constant `CALL_PROXY_IMPL_SLOT` in the `StorageSlots` contract is never used.

Recommendation(s): Consider removing the unused constant.

Status: Unresolved

Comment from the client:

6.5 src/starkware/solidity/components/Governance.sol

6.5.1 [Best Practices] Gas Optimization in `_acceptGovernance(...)`

File(s): Governance.sol

Description: The function `_acceptGovernance(...)` loads `gub.candidateGovernor` from storage twice. Since the `require` statement ensures that `msg.sender` is equal to `gub.candidateGovernor`, `msg.sender` which is `calldata` can be used instead of `gub.candidateGovernor` in the following logic to reduce `SLOAD` calls.

```
function _acceptGovernance() internal {
    // The new governor was proposed as a candidate by the current governor.
    GovernanceInfoStruct storage gub = getGovernanceInfo();
    require(msg.sender == gub.candidateGovernor, "ONLY_CANDIDATE_GOVERNOR");

    // Update state.
    acceptNewGovernor(gub.candidateGovernor);
    gub.candidateGovernor = address(0x0);
}
```

Recommendation(s): Consider using `msg.sender` instead of `gub.candidateGovernor` to save gas.

```
-acceptNewGovernor(gub.candidateGovernor);
+acceptNewGovernor(msg.sender);
```

Status: Unresolved

Comment from the client:

6.6 src/starkware/solidity/upgrade/Proxy.sol

6.6.1 [Medium] Proxy may not accept message cancellation refunds

File(s): Proxy.sol

Description: The Ethereum bridge proxy is designed to reject Ether transfers. Depending on how message cancellation refunds from the StarkNet core contract are implemented, refunds may not be possible. For example, if the StarkNet core contract refunds by sending Ether directly (without calldata) then the transaction will revert due to the `revert` statement in the proxy's `receive(...)` function. The code is shown below.

```
receive() external payable {
    revert("CONTRACT_NOT_EXPECTED_TO_RECEIVE");
}
```

Recommendation(s): As the message cancellation refund mechanism is not implemented yet, flexibility is the best approach. Consider creating a function `handleReceive(...)` in the implementation contract, which is delegate-called by the Proxy on a call to `receive(...)`. In the case that the functionality of receiving needs to change, this can now be done through an implementation upgrade

Status: Unresolved

Comment from the client:

6.6.2 [Low] Unable to upgrade contract without revealing new implementation

File(s): Proxy.sol

Description: When adding an implementation through the function `addImplementation(...)` there is a check to ensure that the argument `newImplementation` is a deployed contract. This means that in order to upgrade an implementation the new implementation must be deployed and therefore publicly known. Combining this requirement with the time delay between proposing and upgrading to an implementation leads to a situation where it is impossible to fix an issue without revealing the fix first. In the case that an exploit is found in the protocol, the patch must be revealed through `addImplementation(...)` and during the time delay before calling `upgradeTo(...)` an attacker may be able to inspect the patched implementation to identify and exploit the issue.

Recommendation(s): Consider removing the requirement that the new implementation must be a deployed contract when calling `addImplementation(...)`. A commitment to the bytecode of the new implementation can also be added to ensure that the new implementation code was not changed during the delay time.

Status: Unresolved

Comment from the client:

6.6.3 [Best Practices] Function `implementationIsFrozen` returns `false` when no implementation is set

File(s): Proxy.sol

Description: The function `implementationIsFrozen(...)` returns `false` when no implementation is set. When the function returns `false` the meaning can be ambiguous. It may mean that the implementation exists and is frozen, or instead that no implementation has been set.

Recommendation(s): When an implementation has not been set, it should not be possible to determine information about its state. Consider reverting when calling this function while the implementation has not been set.

Status: Unresolved

Comment from the client:

6.7 src/starkware/solidity/libraries/NamedStorage.sol

6.7.1 [Info] Function `setAddressValueOnce()` can overwrite values

File(s): NamedStorage.sol

Description: The function `setAddressValueOnce()` does a check to ensure that a value at the given storage slot `tag_` has not already been written to. This check is done with the line `require(getAddressValue(tag_) == address(0x0), "ALREADY_SET")`, however `getAddressValue(...)` will only return the last 20 bytes in the storage slot. If some value was already stored in that slot that had 20 leading `0` bits then that value would be overwritten. This finding is more focused on the library itself, as the way the library is used by the bridge implementation does not lead to a situation where this is possible.

Recommendation(s): Use `getUintValue(...)` for checking if the value in the specified slot is zero which checks the entire 32 bytes rather than `getAddressValue(...)` which only checks 20 bytes.

Status: Unresolved

Comment from the client:

6.7.2 [Info] Gas saving opportunity in `NamedStorage` library

File(s): NamedStorage.sol

Description: Functions in the `NamedStorage` library have the argument `string memory tag_`. Since `tag_` is not modified, passing this variable as a `calldata` would save gas when calling functions from the `NamedStorage` library.

Recommendation(s): Consider passing `tag_` as `calldata` instead of `memory` to save gas on function calls.

Status: Unresolved

Comment from the client:

6.8 src/starkware/solidity/components/Governance.sol

6.8.1 [Info] Governance Model invariants are not checked

File(s): src/starkware/solidity/components/Governance.sol

Description: The Governance Model is based on a mapping holding all the governors. The model does not track the number of governors, and so it is not possible to check for broken invariants. For instance, during initialization, the Governance should have zero governors. When adding a new governor, the number of governors should be increased by one. When removing a governor, the number of governors should be decreased by one. The ability to list the Governors can be particularly important when upgrading the contract.

Recommendation(s): Consider some strategy to ensure that invariants hold after each operation. The Enumerable Map is a data structure that can be useful.

Status: Unresolved

Update from the client:

6.9 src/starkware/starknet/apps/starkgate/eth/StarknetTokenBridge.sol

6.9.1 [High] Fees are not returned after canceling a message

File(s): StarknetTokenBridge.sol.

Description: The StarkNet documentation states that fees spent on `L1` to `L2` messages will be refunded if the message is canceled. The function `depositReclaim(...)` is used to complete a message cancellation and receive the fee refund, however the received fees are not returned to the caller. It should be mentioned that the fee refund mechanism is not implemented on the StarkNet core so waiting until the fee refund mechanism is implemented is recommended.

Recommendation(s): When the fee mechanism is implemented in the StarkNet core, update the logic in `depositReclaim(...)` to forward the returned fees to the user.

Status: Unresolved

Comment from the client:

6.9.2 [Info] Governance can change `l2TokenBridge` without restrictions

File(s): StarknetTokenBridge.sol

Description: The storage variable `l2TokenBridge` can be changed by governance without restrictions. In order to withdraw funds from the bridge a message from this StarkNet address must be consumed. This makes the `l2TokenBridge` an important storage variable that can prevent withdrawals if changed to an `L2` address that does not send withdrawal messages to `L1`. Since this is an important address, extra security measures should be taken when making changes to this storage variable.

Recommendation(s): Consider setting the `l2TokenBridge` storage variable only once and prevent future changes. Alternatively if the `l2TokenBridge` address would need to change in the future, consider adding a time delay similar to the proxy implementation upgrade delay.

Status: Unresolved

Comment from the client:

6.10 src/starkware/solidity/interfaces/ProxySupport.sol

6.10.1 [Low] No commitment to code executed in external initializer

File(s): ProxySupport.sol

Description: When adding a new implementation to the proxy there is a commitment done on the new implementation address and in the arguments used for initialization. This allows everyone to check during the delay time what would be the result of the upgrade process. However, in some cases the initialization process involves a `delegatecall` to an external initializer contract, the only information provided about this contract is the address, but the code can be potentially deployed in the moment of initialization executing actions that were not possible to check.

Recommendation(s): When registering a new implementation that will use external initializer contracts, make a commitment to the code that will be executed.

Status: Unresolved

Comment from the client:

6.11 L2 Issues

6.11.1 [Best Practices] No check for caller address

File(s): starkware/*

Description: Differently from the EVM, `caller address` in Starknet can be zero. This would happen when the call is not done from an account address. Because of this it is considered a good practice to check if the result of `get_caller_address` is zero and act in consequence.

Recommendation(s): Check if the result of `get_caller_address` is zero.

Status: Unresolved

Comment from the client:

6.12 src/starkware/starknet/std_contracts/ERC20/ERC20_base.cairo

6.12.1 [Low] Wrong use `assert_le(...)`

File(s): ERC20_base.cairo

Description: The function `assert_le(...)` is used in the `ERC20_initializer(...)` function to check if the argument `decimals` is lower than `256`. However the function `assert_le(a,b)` effectively checks if `(b - a) < 2^128`. It is possible to pass some values which overflow on `b - a` where the result is still less than `2^128`.

Recommendation(s): Use `assert_le_felt(...)` instead.

Status: Unresolved

Comment from the client:

6.13 src/starkware/starknet/std_contracts/ERC20/ERC20.cairo

6.13.1 [Best Practices] Returning number instead of a boolean

File(s): ERC20.cairo

Description: There are functions in the Cairo contracts that return `1` which represent `TRUE`. However, it is considered good practice to use the StarkWare boolean library allowing the use of the constants `TRUE` and `FALSE` to increase code readability.

Recommendation(s): Consider using the StarkWare boolean library constants to represent true/false values instead of `0`/`1`.

Status: Unresolved

Comment from the client:

6.14 src/starkware/starknet/std_contracts/upgradability_proxy/proxy.cairo

6.14.1 [Info] Unused function import

File(s): proxy.cairo

Description: The contract `proxy.cairo` imports the function `only_governor(...)`, which is not used in the contract.

Recommendation(s): Consider removing the imported `only_governor(...)` function.

Status: Unresolved

Comment from the client:

6.14.2 [Best Practices] Unchecked `final` boolean argument in `upgrade_to(...)`

File(s): proxy_impl.cairo

Description: The function `upgrade_to(...)` contains the argument `final` which is assumed to be boolean. The `final` argument is not validated to be within the range of `[0,1]` however. Additionally, the function `process_final_flag(...)` which does the check to finalize the proxy only checks that the `final` flag is `FALSE`. Passing any other value would cause finalization. These two conditions create an opportunity where any non-zero value can be passed as the `final` argument allowing for different `implementation_key` outputs calculated from `calc_impl_key(...)` even though when calling `upgrade_to` the results would be identical.

Recommendation(s): Consider checking that the argument `final` is within the range `[0,1]`

Status: Unresolved

Comment from the client:

6.15 src/starkware/starknet/std_contracts/ERC20/ERC20_base.cairo

6.15.1 [Low] Incorrect value for ERC20 constant MAX_DECIMALS

File(s): ERC20_base.cairo

Description: To maintain consistency with the ERC20 standard in Solidity, the Cairo implementation enforces that the storage variable `decimals` must be within the size of a `uint8` which is 8 bits. The maximum decimal value that 8 bits can represent is 255, however the value of `MAX_DECIMALS` is set to 256 which is 9 bits (`100000000`). The code is shown below:

```
const MAX_DECIMALS = 256;
...
func ERC20_initializer{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}(
    name: felt, symbol: felt, decimals: felt
) {
    ////////////////////////////////////////////////////
    // @audit-issue "decimals" can be 9 bits
    ////////////////////////////////////////////////////
    assert_le(decimals, MAX_DECIMALS);
    ERC20_name.write(name);
    ERC20_symbol.write(symbol);
    ERC20_decimals.write(decimals);
    return ();
}
```

Recommendation(s): Change `MAX_DECIMALS` to 255.

Status: Unresolved

Comment from the client:

7 Concerns

This section aims to highlight some concerns that the Nethermind team has about the StarkGate bridge protocol. While Section 6 lists clear actionable issues, this focuses on particular design choices that add some level of risk to the bridge.

7.1 Complex proxy implementation

The proxy implementation used for the bridge introduces two concepts: a) EICs (External Initializer Contracts); and b) Subcontracts. There was no documentation for these features, so Nethermind understood these concepts only through reading the code, inline comments and discussions with the StarkWare team.

EICs allow a contracts initialization code to reside in an external contract (hence the name External Initializer Contract) rather than the more common approach of using an initializer function inside the implementation. The StarkWare team has stated that they intend to complete the first initialization without an EIC (rely on the implementation's initializer), but all following implementation upgrades will use an EIC. This poses a limitation in the audit as the codebase supports the EIC logic, but there are no contracts that make use of an EIC so we have not been able to observe it in practice. There are two ways to initialize a new implementation, using the implementations initialize function or using an EIC. For each of these approaches they behave differently depending on whether the contract has already been initialized. This creates four branching paths which adds complexity to the proxy setting/upgrading implementation process.

Subcontracts are supported by the proxy and implementation logic, but similar to EICs they are not used by any contracts in the codebase. Therefore we have only been able to assess the use of subcontracts based on the supporting logic but have not been able to observe it in practice. Implementations have a function `processSubContractAddresses(...)`, but these are left empty for the bridge implementations that we have been provided. With no developer documentation describing this feature and few inline comments, the purpose of subcontracts is unclear. This feature adds further complexity to the upgrade process, where many calls and `delegatecall` to separate contracts may be executed during an implementation upgrade.

In summary, complexity leads to more potential risk. We believe that the approach that the StarkWare team has taken toward proxies is unnecessarily complex and could be simplified to reduce risk.

7.2 Inconsistencies between layers

The governance and proxy implementations for both L1 and L2 are very similar, but there are some minor differences in their implementation. These inconsistencies are outlined below.

- The L1 proxy initializes governance in its constructor during deployment, but the L2 proxy must have its governance initialized in a separate call.
- The L1 governance only supports one governor candidate at a time, but the L2 governance supports multiple candidates being proposed at the same time.
- The L1 `removeImplementation` function will revert when removing a non-existent implementation, but the L2 `remove_implementation` function will silently succeed.

These inconsistencies are not a security issue, but they may cause governors to assume that behavior of governance and proxies are consistent between layers where they may not be, which could lead to false assumptions and unintended actions as a result.

7.3 Governance lacks voting

The governance implementation does not feature any voting mechanisms. Instead, each governor can complete any action without approval by other governors. Some governor-only functions have a time delay to prevent potentially malicious action from occurring, such as the time delay when adding a new implementation. Other actions do not have this time delay however, allowing a malicious governor to immediately make changes to the protocol. An example of such an action is removing governors. If a malicious party gets access to governance, they could remove all other governors immediately which would give them full control over the governance. Voting and requiring an approval threshold could prevent individual governors from executing actions that may harm the protocol. It should be noted that the StarkGate team had stated during a discussion that they plan to only have one governor appointed at a time, and the governance feature exists to swap out governors when needed. An alternative approach could be to have a single owner role and a propose-claim process for setting a new owner, which would eliminate the concern of multiple governors without a voting process.

8 Documentation Evaluation

Technical documentation is created to explain what the software product does. This way, developers and stakeholders can easily follow the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a `README.md` but also using code as documentation (to write clear code), diagrams, websites, research papers, videos and external documentation. Besides being a good programming practice, proper technical documentation improves the efficiency of audits. Less time can be spent understanding the protocol and more time can be put towards auditing which improves the efficiency and overall output of the audit.

The codebase contains a `README` file with information describing the directory layout and how to run tests. However, we were not able to run the tests following the provided instructions. There are also links to docs.starknet.io providing an overview of how the bridge works. While the `README` and links are useful, the code would benefit from detailed developer documentation.

We recommend the following improvements be made to the documentation of this project:

- Increase the line comments for Cairo contracts: while the Solidity contracts have a ratio of 46.5% of comments, the Cairo contracts have 9.5%;
- Improve the architecture description of the system;
- Improve the technical specification of the project, detailing each use cases;
- Formalize the functional and non-functional requirements of the project.

9 Test Suite Evaluation

9.1 Contracts Compilation Output

The documentation didn't provide instructions on how to compile the project.

9.2 Tests Output

The `Dockerfile` provided didn't executed successfully.

9.3 Code Coverage

Code coverage is a software testing metric that determines the number of lines of code that is successfully validated under a test procedure, which in turn, helps in analyzing how comprehensively a software is verified. **This project is not instrumented with code coverage**.

10 Static Analysis Tools

As part of our review, we run Slither and Amarna over the contracts. These are static analyzers that help us to highlight common issues on Smart Contracts written in Solidity and Cairo respectively. Even if we decide to not highlight some of these issues or downgrade severity of others we strongly recommend to the Starkware team to conduct their own evaluation of this analysis. Below we present the results of these tools on each contract

10.1 Slither

Slither was used over the Solidity files, the next command was used for running Slither:

```
cd src && slither $path_to_file_from_src --solc-args "--allow-paths $(pwd)"
```

10.1.1 src/starkware/starknet/apps/starkgate/eth/StarknetERC20Bridge.sol

For this file, the tool highlighted 1 **Critical**, 5 **Medium**, 4 **Low** and 29 **Informational** issues. The **Critical** issue is related to the use of `delegatecall` over an address received as input. This issue is referenced in the finding 6.10.1, we decided to decrease severity because it is only accessible by governance and after the set delay has passed.

Two of the **Medium** issues are related to strict equality checks on the `Transfers` library. We considered these do not cause a vulnerability in this scenario, transfers are done from the same contract and this defines part of the expected behavior for tokens. The other three issues are related to unused return values, the unused return value is the hash of the messages sent to StarkNet, in the current implementation this is ignored because the nonce of the message is used as a unique identifier.

The **Low** and **Informational** issues are mostly related to "external calls", "event emitted after calls", "unused functions", "use of inline assembly", and "use of low level calls". After reviewing the code and conversations with StarkWare, we decide to not highlight independently any of these issues.

10.1.2 src/starkware/starknet/apps/starkgate/eth/StarknetEthBridge.sol

The tool highlighted 1 **Critical**, 3 **Medium**, 4 **Low** and 30 **Informational** issues. The **Critical** and **Medium** issues highlighted on this file, were also highlighted in the previous file, so we kept the same decisions on this scenario. In the case of **Low** and **Informational** issues they remain in the same categories and we also decide to not highlight any of them individually.

10.1.3 src/starkware/solidity/upgrade/Proxy.sol

For this file, the tool highlighted 1 **Critical**, 0 **Medium**, 2 **Low** and 20 **Informational** issues. The **Critical** issue refers to a uninitialized storage variable. This variable will be initialized in the initGovernance function that will be called in the initialization process. The **Low** and **Informational** issues are mostly related to "external calls", "event emitted after calls", "unused functions", "use of inline assembly", and "use of low level calls". After reviewing the code and conversations with Starkware, we decide to not highlight independently any of these issues.

10.2 Amarna

Amarna was used over the Cairo files, the next command was used for running Amarna:

```
amarna $path_to_file -s
```

10.2.1 src/starkware/starknet/apps/starkgate/cairo/token_bridge.cairo

For this file, the tool highlighted 2 **Warning**, and 4 **Info** issues. Two of these issues are related to non checked overflows, we consider these false positives because those operations are indeed checked for overflows in the next lines. Other two of these issues are unused imports. One of these correctly points to `assert_lt_felt` as not used. The other one is a false positive on the view function `initialized`. The last two issues are related to not checking the "caller address". This is mentioned in the finding 6.11.1.

10.2.2 src/starkware/starknet/std_contracts/ERC20/ERC20.cairo

For this file, the tool highlighted 1 **Warning**, and 14 **Info** issues. The **Warning** issues refer to an unchecked overflow, we consider this as a false positive because the overflow value is checked in the next line. Three of the **Info** issues are related to the use of native arithmetic operations in Cairo, we also consider these as false positives and the specified use of arithmetic operations is correct. From the remaining **Info** issues, six of them refer to unused imports, we consider only one of them, referring to the storage variable `permitted_minter` is valid. The other five issues refer to not checking caller address, which is mentioned in the finding 6.11.1.

10.2.3 src/starkware/starknet/std_contracts/ERC20/ERC20_base.cairo

For this file, the tool highlighted 3 **Warning**, and 6 **Info** issues. The **Warning** issues are about unchecked overflows, we consider them false positives, one of these cases is actually checked in the next line for overflows. For the other two issues, the overflow is not possible because of other properties of the system. From the **Info** issues, one of them refers to `SignatureBuiltin` as an unused import, we agree with this. The other five refers to unused functions. We consider these false positives because the specified functions are used in other files.

10.2.4 src/starkware/starknet/std_contracts/ERC20/permitted.cairo

For this file, the tool highlighted 3 **Info** issues. Two of the highlighted issues refer to unused functions, we consider these false positives because those functions are used in other files. The last issue is related to not checking the caller address, which is mentioned in the finding 6.11.1.

10.2.5 src/starkware/starknet/std_contracts/upgradability_proxy/finalizable.cairo

For this file, the tool highlighted 1 **Info** issue referring to an unused function. We consider this a false positive because this function is meant to be used in other contracts.

10.2.6 src/starkware/starknet/std_contracts/upgradability_proxy/governance.cairo

For this file, the tool highlighted 6 **Info** issues related to not checking the caller address. This is mentioned in 6.11.1.

10.2.7 src/starkware/starknet/std_contracts/upgradability_proxy/initializable.cairo

For this file, the tool highlighted 1 **Info** issue referring to an unused function. We consider this a false positive because this function is meant to be used in other contracts.

10.2.8 src/starkware/starknet/std_contracts/upgradability_proxy/proxy.cairo

For this file, the tool highlighted 13 **Info** issues. All the issues refer to unused imports. From these we consider only the issue referring to `only_governor` is valid, the rest of the issues specify externally visible functions.

10.2.9 src/starkware/starknet/std_contracts/upgradability_proxy/proxy_impl.cairo

For this file, the tool highlighted 2 **Warning** and 2 **Info** issues. The two **Warning** issues are related to unused arguments. The unused arguments are `pedersen_ptr` and `range_check_ptr` from the function `initialize`, we consider this is correct, but not relevant enough for being independently highlighted. One of the **Info** issues refers to the use of a native arithmetic function, which is correct. The other **Info** issue refers to an unused import, specifically to the value `TRUE`, we consider this is correct and this import could be removed.

11 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.