StarkGate Security Review

NETHERMIND

Reviewers
Greg Vardy, Lead
Mauricio Perdomo, Lead
April 5, 2022

1 Executive Summary

Over the course of 14 days in total, Starkware engaged with Nethermind to review StarkGate.

We found a total of 5 issues with StarkGate. Our main concerns are related to users not having an emergency mechanism for unlocking their funds if Starknet became unresponsive or starts to censor transactions. Beside the issues mentioned we added a set of notes and recommendations that, from our point of view, would improve code and protocol quality.

Repository	**Commit**
StarkGate	403136096b14eff95ddee77795322108991c598a

Summary

Type of Project	Rollup Bridge
Timeline	Mar 7, 2022 - Mar 24, 2022
Methods	Manual Review
Documentation	Medium

Total Issues

High Risk	0
Medium Risk	3
Low Risk	2
Notes and recommendations	18

Contents

2 Nethermind

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.

3 Introduction

StarkGate is a token bridge created by Starkware for transferring assets from Ethereum L1 to Starknet and from Starknet to Ethereum L1. The bridges facilitate a 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 even ETH and ERC20 tokens never actually leave Ethereum.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a two-person team. Any modifications to the code will require a new security review.

4 Findings

4.1 Medium Risk

4.1.1 Funds locked if Starknet became unresponsive or censorship.

Severity: Medium

Context: `StarknetTokenBridge.sol`

Description After users transfers their funds to Starknet, they must call the `initiate_withdraw` function to unlock their funds on the L1 again. If Starknet become unresponsive or censors some users, making them unable to call the `initiate_withdraw` function, those users will not be able to unlock their funds.

Recommendation Add documentation to make this risk explicit to the user. Create a mechanism to allow users to unlock their funds if Starknet becomes

unresponsive or censors them.

4.1.2 Uint256 not validated.

Severity: Medium

Context: token_bridge.cairo#L187, token_bridge#L228

Description In the `initiate_withdraw` function, `amount` is used to call the `permissionedBurn` function and to create the message for unlocking funds on L1. Not checking this value could bring errors during the execution of `permissionedBurn` function. In the `handle_desposit` function, `low` and `high` values are assumed to be in the correct ranges. If they are not, this could result in errors during the call to the `permissionedMint` function. This is not currently a problem, because values are checked within the actual implementation of tokens. Also, for `permissionedMint` `low` and `high` values are crafted correctly in the L1 part of the bridge, but this could change in future. The decision to make these checks in the called functions is correctly documented.

```
@external
func initiate_withdraw{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
 ↪  range_check_ptr}(
        l1_recipient : felt, amount : Uint256):
   with_attr error_message("RECIPIENT_ADDRESS_OUT_OF_RANGE"):
        assert_lt_felt(l1_recipient, ETH_ADDRESS_BOUND)
    end

    # Call burn on l2_token contract.
    let (caller_address) = get_caller_address()
    let (l2_token_) = get_l2_token()
    with_attr error_message("UNINITIALIZED_TOKEN"):
        assert_not_zero(l2_token_)
    end
    IMintableToken.permissionedBurn(
        contract_address=l2_token_, account=caller_address, amount=amount)
    ......
```

```
@l1_handler
func handle_deposit{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}(
        from_address : felt, account : felt, amount_low : felt, amount_high :
        ↪  felt):
    # The amount is validated (i.e. amount_low, amount_high < 2**128) by an
    ↪  inner call to
    # IMintableToken permissionedMint function.

    let (expected_from_address) = get_l1_bridge()
    with_attr error_message("EXPECTED_FROM_BRIDGE_ONLY"):
        assert from_address = expected_from_address
    end
    let amount : Uint256 = cast((low=amount_low, high=amount_high), Uint256)

    # Call mint on l2_token contract.
    let (l2_token_) = get_l2_token()
    with_attr error_message("UNINITIALIZED_TOKEN"):
        assert_not_zero(l2_token_)
    end
    IMintableToken.permissionedMint(contract_address=l2_token_,
    ↪  account=account, amount=amount)
    ......
```

Recommendation: Check that values are in the correct ranges in the `initiate_withdraw` and `handle_desposit` functions.

```
with_attr error_message("RECIPIENT_ADDRESS_OUT_OF_RANGE"):
    assert_lt_felt(l1_recipient, ETH_ADDRESS_BOUND)
end
+ with_attr error_message("INVALID AMOUNT"):
+    uint256_check(amount)
+ end
...
```

```
let amount : Uint256 = cast((low=amount_low, high=amount_high), Uint256)
+ with_attr error_message("INVALID AMOUNT"):
+    uint256_check(amount)
+ end
...
```

4.1.3 Effects of external functions are not checked.

Severity: Medium

Context: StarknetTokenBridge.sol

Description: Calls to the `initiate_withdraw` or `handle_deposit` functions result in calls to the `permissionedBurn` and `permissionedMint` functions respectively. The cairo bridge expect these functions to burn or mint the correct amount of tokens and revert in case of an error. This may lead to a vulnerability similar to what happens when an ERC20 tokens returns `false` in some of its functions instead of revert. If these calls fail silently, the results would be tokens in Starknet without collateral or funds locked on the L1.

```
@external
func initiate_withdraw{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}(
        l1_recipient : felt, amount : Uint256):
   with_attr error_message("RECIPIENT_ADDRESS_OUT_OF_RANGE"):
        assert_lt_felt(l1_recipient, ETH_ADDRESS_BOUND)
    end

    # Call burn on l2_token contract.
    let (caller_address) = get_caller_address()
    let (l2_token_) = get_l2_token()
    with_attr error_message("UNINITIALIZED_TOKEN"):
        assert_not_zero(l2_token_)
    end
    IMintableToken.permissionedBurn(
        contract_address=l2_token_, account=caller_address, amount=amount)
    ......
```

```
@l1_handler
func handle_deposit{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}(
        from_address : felt, account : felt, amount_low : felt, amount_high :
        ↪  felt):
    # The amount is validated (i.e. amount_low, amount_high < 2**128) by an
    ↪  inner call to
    # IMintableToken permissionedMint function.

    let (expected_from_address) = get_l1_bridge()
    with_attr error_message("EXPECTED_FROM_BRIDGE_ONLY"):
        assert from_address = expected_from_address
    end
    let amount : Uint256 = cast((low=amount_low, high=amount_high), Uint256)

    # Call mint on l2_token contract.
    let (l2_token_) = get_l2_token()
    with_attr error_message("UNINITIALIZED_TOKEN"):
        assert_not_zero(l2_token_)
    end
    IMintableToken.permissionedMint(contract_address=l2_token_,
    ↪  account=account, amount=amount)
    ......
```

Recommendation: Check that the correct amount was added or substracted respectively from the account balance.

```
+ let (balance_before) = IERC20.balanceOf(contract_address=l2_token_,
↪  account=account)
IMintableToken.permissionedBurn(
        contract_address=l2_token_, account=caller_address, amount=amount)
+ let (balance_after) = IERC20.balanceOf(contract_address=l2_token_,
↪  account=account)

+ let (balance_change) = uint256_sub(balance_before, balance_after)
+ with_attr error_message("WRONG BALANCE CHANGE"):
+    assert uint256_eq(balance_change, amount)
+ end
...
```

```
+ let (balance_before) = IERC20.balanceOf(contract_address=l2_token_,
↪  account=account)
IMintableToken.permissionedMint(contract_address=l2_token_, account=account,
↪  amount=amount)
+ let (balance_after) = IERC20.balanceOf(contract_address=l2_token_,
↪  account=account)

+ let (balance_change) = uint256_sub(balance_after, balance_before)

+ with_attr error_message("WRONG BALANCE CHANGE"):
+    assert uint256_eq(balance_change, amount)
+ end
...
```

4.2 Low Risk

4.2.1 Zero address recipient on Starknet withdraw.

Severity: Low

Context: token_bridge.cairo#L170-L209

Description: `initiate_withdraw` can be called with zero as the recipient address when calling the `withdraw` function on the L1 part of the bridge. This would result in eth being burnt, or the tokens being permanently locked because `transferOut` checks that `recipient != address(0)`.

```
@external
func initiate_withdraw{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}(
        l1_recipient : felt, amount : Uint256):
   with_attr error_message("RECIPIENT_ADDRESS_OUT_OF_RANGE"):
        assert_lt_felt(l1_recipient, ETH_ADDRESS_BOUND)
    end
    ...
```

Recommendation: Check that the `l1_recipient` argument in the `initiate_withdraw` function is not zero.

```
...
with_attr error_message("RECIPIENT_ADDRESS_OUT_OF_RANGE"):
    assert_lt_felt(l1_recipient, ETH_ADDRESS_BOUND)
end

+with_attr error_message("RECIPIENT_ZERO_ADDRESS")
+ assert_not_zero(l1_recipient)
+end
...
```

4.2.2 Zero address recipient on Starknet deposit.

Severity: Low

Context: token_bridge.cairo#L211-L232

Description: `handle_desposit` doesn't check if the account receiving the assets is zero or not, this could lead to funds being burnt. This is checked in the L1 part of the bridge and also in the `mint` function from the ERC20 token in Starknet, but the fact that this is checked in other places it is not documented.

```
@l1_handler
func handle_deposit{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}(
        from_address : felt, account : felt, amount_low : felt, amount_high :
        ↪  felt):
    # The amount is validated (i.e. amount_low, amount_high < 2**128) by an
    ↪  inner call to
    # IMintableToken permissionedMint function.

    let (expected_from_address) = get_l1_bridge()
    with_attr error_message("EXPECTED_FROM_BRIDGE_ONLY"):
        assert from_address = expected_from_address
    end

    let amount : Uint256 = cast((low=amount_low, high=amount_high), Uint256)

    # Call mint on l2_token contract.
    let (l2_token_) = get_l2_token()
    with_attr error_message("UNINITIALIZED_TOKEN"):
        assert_not_zero(l2_token_)
    end
    IMintableToken.permissionedMint(contract_address=l2_token_,
    ↪  account=account, amount=amount)
    ...
```

Recommendation: Document the check that the account receiving the funds is not zero. Add a check to ensure it is not.

```
...
let (expected_from_address) = get_l1_bridge()
with_attr error_message("EXPECTED_FROM_BRIDGE_ONLY"):
    assert from_address = expected_from_address
end

+ with_attr error_message("ACCOUNT_ZERO_ADDRESS")
+   assert_not_zero(account)
+ end
...
```

4.3 Notes and recommendations

4.3.1 decreaseAllowance doesn't allow zero amount.

Context: ERC20.cairo#L122

Description: When `decreaseAllowance` is called with `substracted_value` equal to zero, the new allowance will be equal to the previous one making the transaction fail because this is checked using the strictly less than operator. Common ERC20 implementations using `decreaseAllowance` allow the allowance to be reduce by zero, therefore, this may result in a semantic change that could cause issues with other contracts integrating with this one.

```
...
let (enough_allowance) = uint256_lt(new_allowance, current_allowance)
assert_not_zero(enough_allowance)
...
```

Recommendation: Use the `uint256_le` function instead of the `uint256_lt` function to allow the allowance to be reduced by zero.

```
...
- let (enough_allowance) = uint256_lt(new_allowance, current_allowance)
+ let (enough_allowance) = uint256_le(new_allowance, current_allowance)
...
```

4.3.2 Multiple implementation of the `only_governor` check.

Context: token_bridge.cairo#L118-L122, token_bridge.cairo#L145-L168

Description: Both functions `set_l1_bridge` and `set_l2_token` use the same snippet of code to check if the caller is the governor. This could lead to divergence in future implementations.

```
@external
func set_l1_bridge{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
 ↪  range_check_ptr}(
        l1_bridge_address : felt):
    # The call is restricted to the governor.
    let (caller_address) = get_caller_address()
    let (governor_) = get_governor()
    with_attr error_message("GOVERNOR_ONLY"):
        assert caller_address = governor_
    end

...

@external
func set_l2_token{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
 ↪  range_check_ptr}(
        l2_token_address : felt):
    # The call is restricted to the governor.
    let (caller_address) = get_caller_address()
    let (governor_) = get_governor()
    with_attr error_message("GOVERNOR_ONLY"):
        assert caller_address = governor_
    end
...
```

Recommendation: We recommend that this snippet be put into a dedicated function called `only_governor` and used in both functions.

```
+func only_governor():
+    let (caller_address) = get_caller_address()
+    let (governor_) = get_governor()
+    with_attr error_message("GOVERNOR_ONLY"):
+        assert caller_address = governor_
+    end
+end

...

@external
func set_l1_bridge{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪   range_check_ptr}(
        l1_bridge_address : felt):
    # The call is restricted to the governor.
-    let (caller_address) = get_caller_address()
-    let (governor_) = get_governor()
-    with_attr error_message("GOVERNOR_ONLY"):
-        assert caller_address = governor_
-    end
+    only_governor()

...

@external
func set_l2_token{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪   range_check_ptr}(
        l2_token_address : felt):
    # The call is restricted to the governor.
-    let (caller_address) = get_caller_address()
-    let (governor_) = get_governor()
-    with_attr error_message("GOVERNOR_ONLY"):
-        assert caller_address = governor_
-    end
+    only_governor()

...
```

4.3.3 No error messages in assertions.

Context: ERC20_base.cairo

Description: A lot of `assert` statements in the file do not use a message to describe the error, this could result in bad user experiences.

Recommendation: We recommend adding descriptive messages to the asser-

tions.

4.3.4 Naming of storage variable arguments relating to governance could be more descriptive.

Context: governance.cairo#L15, governance.cairo#L19

Description: The naming of the arguments to the `governors` and `candidates` storage variables is not very descriptive. Because in cairo there is not a boolean type, a felt having value 0 or 1 is used to represent one. Therefore, the type of a storage variable used to store whether or not a specific address satisfies a specific condition would be `felt -> felt`. This type does not express to a developper using it, that it is meant to return a boolean, this could be mitigated using changes in the naming of the return variables.

```
...

@storage_var
func governors(account : felt) -> (active_governor : felt):
end

@storage_var
func candidates(account : felt) -> (governance_candidate : felt):
end

...
```

Recommendation: Use names like: `is_governor`, `is_active_governor`, `is_candidate` for the return value.

```
...

@storage_var
-func governors(account : felt) -> (active_governor : felt):
+func governors(account : felt) -> (is_active_governor : felt):
end

@storage_var
-func candidates(account : felt) -> (governance_candidate : felt):
+func candidates(account : felt) -> (is_governance_candidate : felt):
end

...
```

4.3.5 Not specific event emitted when `init_governance` is called.

Context: governance.cairo#L50-L62

Description: When `init_governance` is called, there is not specific `init` event to emit, instead a sequence of `governor_nominated` and `governance_accepted` is emitted. This could be insufficiently specific for an off-chain tool.

```
@external
func init_governance{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}():
    let (already_init : felt) = governance_initialized.read()
    with_attr error_message("ALREADY_INITIALIZED"):
        assert already_init = FALSE
    end
    let (caller : felt) = get_caller_address()
    governance_initialized.write(TRUE)
    governors.write(account=caller, value=TRUE)
    governor_nominated.emit(new_governor_nominee=caller, nominated_by=caller)
    governance_accepted.emit(new_governor=caller)
    return ()
end
```

Recommendation: We recommend adding an extra event `governance_initialized(governor : felt)` to be emitted when `init_governance` is successfully called.

```
...
+@event
+func governance_initialized(governor : felt)
+end
...


@external
func init_governance{syscall_ptr : felt*, pedersen_ptr : HashBuiltin*,
↪  range_check_ptr}():
    let (already_init : felt) = governance_initialized.read()
    with_attr error_message("ALREADY_INITIALIZED"):
        assert already_init = FALSE
    end
    let (caller : felt) = get_caller_address()
    governance_initialized.write(TRUE)
    governors.write(account=caller, value=TRUE)
-    governor_nominated.emit(new_governor_nominee=caller, nominated_by=caller)
-    governance_accepted.emit(new_governor=caller)
+    governance_initialized(caller)
    return ()
end
```

4.3.6 Use of `assert_not_zero` instead of `assert var = TRUE`.

Context: governance.cairo#L127

Description:

`is_candidate` always has values FALSE (0) or TRUE (1) and checking `assert_not_zero(is_candidate)` is more expensive than checking `assert is_candidate = TRUE`.

```
...
with_attr error_message("NOT_A_GOVERNANCE_CANDIDATE"):
        assert_not_zero(is_candidate)
end
...
```

Recommendation: Use `assert is_candidate = TRUE`.

```
...
with_attr error_message("NOT_A_GOVERNANCE_CANDIDATE"):
-         assert_not_zero(is_candidate)
+       assert is_candidate = TRUE
end
...
```

4.3.7 Naming inconsistency.

Description: There is no consistent naming convention throughout the whole codebase. For instance, sometimes internal functions start with an underscore, and sometimes they do not. This results in less readable code in which issues can arise more easily.

Recommendation: Keep a consistent naming convention and code style. For example, the one defined in the Solidity documentation.

4.3.8 Typos in values used for computing hashes.

Context: `StorageSlots.sol`

Description: These values are the keccak256 hashes of specific strings, some of these strings contain the word "implemntation", which appears to be a typo.

4.3.9 StarknetTokenBridge inherits ContractInitializer twice.

Context: `StarknetTokenBridge.sol#L11`

Description: `StarknetTokenBridge` inherits `ContractInitializer` twice, once through `ProxySupport`, as well as directly. This is not an issue but could make subclassing and the type hierarchy a little more complex.

```
...
abstract contract StarknetTokenBridge is
    StarknetTokenStorage,
    GenericGovernance,
    ContractInitializer,
    ProxySupport
{
...
```

Recommendation: It would seem that `StarknetTokenBridge` could inherit `ContractInitializer` only through `ProxySupport`.

```
...
abstract contract StarknetTokenBridge is
    StarknetTokenStorage,
    GenericGovernance,
-    ContractInitializer,
    ProxySupport
{
...
```

4.3.10 Unnecesary `l2TokenBridgeNotSet` modifier.

Context: StarknetTokenBridge#L65

Description: The `l2TokenBridgeNotSet` modifier is only used in the `setL2TokenBridge` function but appears to be unnecesary because it sets the value of `l2TokenBridge` by calling `setUintValueOnce` which already checks if the values was previously set.

```
...

modifier l2TokenBridgeNotSet() {
        require(l2TokenBridge() == 0, "L2_TOKEN_CONTRACT_ALREADY_SET");
        _;
    }

...

function setL2TokenBridge(uint256 l2TokenBridge_)
        external
        l2TokenBridgeNotSet
        isValidL2Address(l2TokenBridge_)
        onlyGovernance
    {
        emit LogSetL2TokenBridge(l2TokenBridge_);
        l2TokenBridge(l2TokenBridge_);
    }

...
```

Recommendation: Remove `l2TokenBridgeNotSet` modifier.

```
- modifier l2TokenBridgeNotSet() {
-         require(l2TokenBridge() == 0, "L2_TOKEN_CONTRACT_ALREADY_SET");
-         _;
-     }

...

 function setL2TokenBridge(uint256 l2TokenBridge_)
         external
-         l2TokenBridgeNotSet
         isValidL2Address(l2TokenBridge_)
         onlyGovernance
     {
         emit LogSetL2TokenBridge(l2TokenBridge_);
         l2TokenBridge(l2TokenBridge_);
     }
```

4.3.11 Deposit is not part of the interface of StarknetTokenBridge.

Context: StarknetTokenBridge.sol

Description: StarknetTokenBridge defines a `withdraw` method as part of its interface, but it doesn't define `deposit`. We think both method should be part of the interface of this contract.

Recommendation: Add the declaration of the `deposit` method to the StarknetTokenBridge contract.

4.3.12 Expensive operations are done before checking requirements.

Context: StarknetEthBridge.sol#L16-22

Description: The `withdraw` method call invokes `consumeMessage` before checking certain requirements, this could cause `consumeMessage` to execute unnecessarily when requirements are not met. If these requirements are checked first, less gas will be used when the requirements do not hold.

```
...
function withdraw(uint256 amount, address recipient) public override {
        consumeMessage(amount, recipient);
        // Make sure we don't accidentally burn funds.
        require(recipient != address(0x0), "INVALID_RECIPIENT");
        require(address(this).balance - amount <= address(this).balance,
↪   "UNDERFLOW");
        recipient.performEthTransfer(amount);
    }
...
```

Recommendation: Check requirements before calling `consumeMessage`.

```
...
 function withdraw(uint256 amount, address recipient) public override {
+        require(recipient != address(0x0), "INVALID_RECIPIENT");
+        require(address(this).balance - amount <= address(this).balance,
↪   "UNDERFLOW");
        consumeMessage(amount, recipient);
        // Make sure we don't accidentally burn funds.
-        require(recipient != address(0x0), "INVALID_RECIPIENT");
-        require(address(this).balance - amount <= address(this).balance,
↪   "UNDERFLOW");
        recipient.performEthTransfer(amount);
    }
...
```

4.3.13 There is no event emitted when governance is initialized.

Context: `Governance.sol`

Description: `Governance` does not emit an event when it is initialized. The governance contract uses events to give notice of important operations. This could be very useful for off-chain components like monitoring systems, but it does not emit an event upon initialization.

```
   function initGovernance() internal {
        GovernanceInfoStruct storage gub = getGovernanceInfo();
        require(!gub.initialized, "ALREADY_INITIALIZED");
        gub.initialized = true; // to ensure addGovernor() won't fail.
        // Add the initial governer.
        addGovernor(msg.sender);
    }
```

Recommendation: Add a new event `GovernanceInitialized(address)` and

log it when `initGovernance` is called.

```
...
+   event GovernanceInitialized(address governor);
...

    function initGovernance() internal {
            GovernanceInfoStruct storage gub = getGovernanceInfo();
            require(!gub.initialized, "ALREADY_INITIALIZED");
            gub.initialized = true; // to ensure addGovernor() won't fail.
            // Add the initial governer.
            addGovernor(msg.sender);
+           log GovernanceInitialized(msg.sender)
    }
...
```

4.3.14 Know limitation of a method is not documented.

Context: Common.sol#L9

Description: The method `isContract` has known limitations when checking if a specific address is a Smart Contract, these limitations should be documented.

Recommendation: Add comments on the function explaining why this is not always reliable.

4.3.15 Multiple Governors could increase risks.

Description: The governance system used includes multiple governor addresses. Each governor is able to remove any other governor, thus taking control of the governance system. Even though user funds are partially protected because no governor can change the current implementation without a set delay, this would leave the bridge for this specific asset under the control of a malicious actor without any solution. Also, having multiple governors increases the risk of having a governor's private key leaked.

Recommendation: Have only one governor address. This address can be a Smart Wallet allowing multiple kinds of management. There could be addresses allowed to do certain operations without the permission of others, as well as having critical and emergency operations under a voting system. In this way, if a malicious actor takes control of an address with privileges, they can easily be revoked using the Smart Wallet, preventing the system from being taken over.

4.3.16 Use modifiers for doing common and simple checks

Context: StarknetTokenBridge.sol#L86

Description: Modifiers are generally used for doing simples and common checks. `onlyDepositor` could be defined as a modifier.

```
function onlyDepositor(uint256 nonce) internal {
        require(depositors()[nonce] == msg.sender, "ONLY_DEPOSITOR");
}
```

Recommendation: Make `onlyDepositor` a modifier instead of a function.

```
-     function onlyDepositor(uint256 nonce) internal {
+     modifier onlyDepositor(uint256 nonce) {
          require(depositors()[nonce] == msg.sender, "ONLY_DEPOSITOR");
+         _
      }
```

4.3.17 Use of low-level calls

Context: StarknetTokenBridge.sol#L129

Description: Using low-level calls can be error prone, it is better to avoid them when possible.

```
...
(bool success, bytes memory returndata) =
↪   address(messagingContract()).staticcall(
    abi.encodeWithSignature("l1ToL2MessageNonce()")
);
require(success, string(returndata));
uint256 nonce = abi.decode(returndata, (uint256));
...
```

Recommendation: In the function `sendMessage` of `StarknetTokenBridge` contract, a low-level call is used for calling `l1ToL2MessageNonce` in the `messagingContract`. We recommend to add this function to the definition of the `IStarknetMessaging` interface for removing the use of the low-level call.

4.3.18 Expensive operations are done before checking requirements.

Context: StarknetTokenStorage.sol#L184, StarknetTokenStorage.sol#L201

Description: The methods used for the cancellation process do an external call and after return they check if the caller is the correct depositor for cancelling the message. This check could be done before doing the external call, saving gas when the requirement is not met.

```
function depositCancelRequest(
        uint256 amount,
        uint256 l2Recipient,
        uint256 nonce
    ) external {
        messagingContract().startL1ToL2MessageCancellation(
            l2TokenBridge(),
            DEPOSIT_SELECTOR,
            depositMessagePayload(amount, l2Recipient),
            nonce
        );

        // Only the depositor is allowed to cancel a deposit.
        onlyDepositor(nonce);
        emit LogDepositCancelRequest(msg.sender, amount, l2Recipient, nonce);
    }

    function depositReclaim(
        uint256 amount,
        uint256 l2Recipient,
        uint256 nonce
    ) external {
        messagingContract().cancelL1ToL2Message(
            l2TokenBridge(),
            DEPOSIT_SELECTOR,
            depositMessagePayload(amount, l2Recipient),
            nonce
        );

        // Only the depositor is allowed to reclaim cancelled deposit funds.
        onlyDepositor(nonce);
        transferOutFunds(amount, msg.sender);
        emit LogDepositReclaimed(msg.sender, amount, l2Recipient, nonce);
    }
```

Recommendation: Check requirements before calling external functions. This requirements can be implemented as modifiers as recommended before.

```
 function depositCancelRequest(
         uint256 amount,
         uint256 l2Recipient,
         uint256 nonce
     ) external {
+         onlyDepositor(nonce);
         messagingContract().startL1ToL2MessageCancellation(
             l2TokenBridge(),
             DEPOSIT_SELECTOR,
             depositMessagePayload(amount, l2Recipient),
             nonce
         );

-         // Only the depositor is allowed to cancel a deposit.
-         onlyDepositor(nonce);
         emit LogDepositCancelRequest(msg.sender, amount, l2Recipient, nonce);
     }

     function depositReclaim(
         uint256 amount,
         uint256 l2Recipient,
         uint256 nonce
     ) external {
+         onlyDepositor(nonce);
         messagingContract().cancelL1ToL2Message(
             l2TokenBridge(),
             DEPOSIT_SELECTOR,
             depositMessagePayload(amount, l2Recipient),
             nonce
         );

-         // Only the depositor is allowed to reclaim cancelled deposit funds.
-         onlyDepositor(nonce);
         transferOutFunds(amount, msg.sender);
         emit LogDepositReclaimed(msg.sender, amount, l2Recipient, nonce);
     }
```