QuillAudits

Audit Report
July, 2024

For

LIDO STARKNET
GOVERNANCE FORWARDER

Table of Content

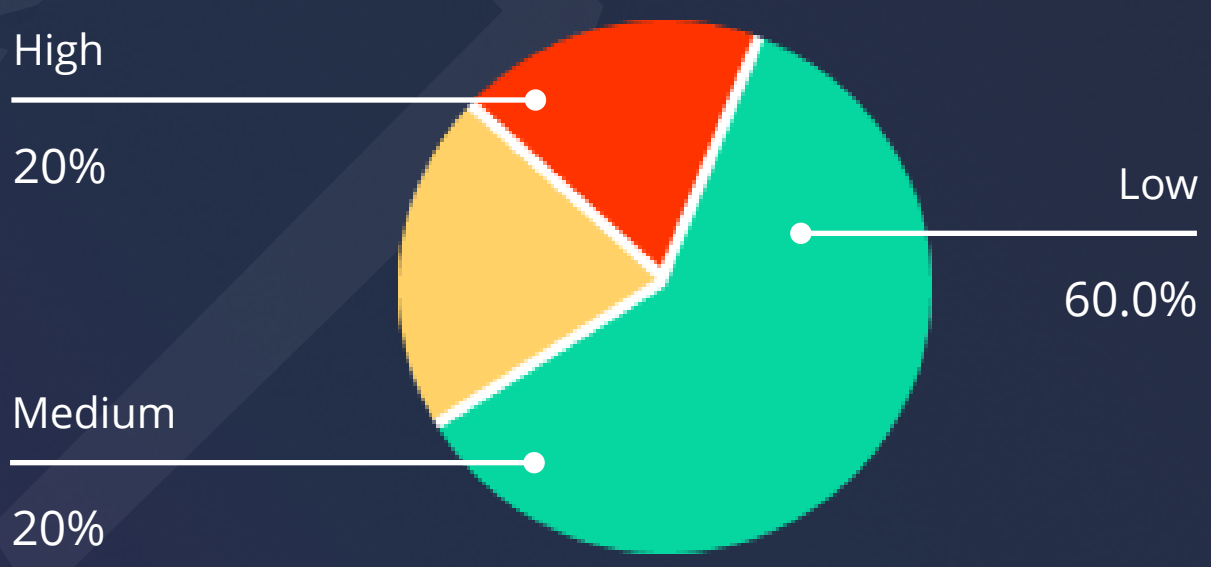
Executive Summary

Project Name	Lido Starknet Governance Forwarder
Overview	The core contract is the BridgeExecutor. This contract contains the logic to facilitate the queueing, delay, and execution of sets of actions on the Starknet network. The contract is implemented to facilitate the execution of arbitrary actions after governance approval on Ethereum.
Timeline	19th June 2024 to 28th June 2024
Update Code Received	23rd July 2024
Second Review	23rd July 2024 - 25th July 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the Lido Starknet Governance Forwarder smart contract's codebase for quality, security, and correctness.
Source Code	https://github.com/0xSpaceShard/lido-starknet-governance-forwarder
Commit Hash	80545985b65e151729cfb9a0b4173a7a102b4e0e
Branch	Main
Fixed In	https://github.com/0xSpaceShard/lido-starknet-governance-forwarder/blob/main/src/bridge_executor/bridge_executor.cairo a4965a1663beb28ca4913aacfcf8775e330c35e4

Number of Security Issues per Severity

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	1	1	0



Checked Vulnerabilities

Standard vulnerabilities checklists, including but not limited to:

- L1-L2 Addresses Conversion
- Transaction-ordering dependence
- Validation of input data
- Business Logic
- Review of Dependencies
- Correct calculations and precision
- Vulnerability for Denial-of-Service (DoS) attacks
- Slippage and Flashloans/Big liquidity vulnerabilities
- Storage issues (uninitialized, unused, etc) and incorrect local variables usage
- Upgradeability issues

and other potential Cairo vulnerabilities and attack vectors;

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC's standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four severity levels, each of which has been explained below.

High Severity Issues

A high severity issue or vulnerability means your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These issues were identified in the initial audit and successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Inefficient _get_current_state view function leads insufficient gas issues during execute function.

Path

src/bridge_executor/bridge_executor.cairo#L356

Function

execute()

Description

In the current implementation of the execute function, it first checks whether the current state of provided action_set_id is in queued state or not using _get_current_state function however _get_current_state() function internally calls _load_actions_set_by_id function which loops through all the mapping and re-generate the ActionSet for the given action_set_id, which is not necessary to know current state of the action_set_id, It can be done using below three storage reads i.e.

```
let has_canceled = self.canceled.read(actions_set_id);
let has_executed = self.executed.read(actions_set_id);
let execution_time = self.execution_time.read(actions_set_id);
```

Because of this design _get_current_state() function is inefficient and charges high cost for overall execute function execution which has a possibility of insufficient gas errors if more than couple of calls are executed within the provided action_set_id.

Same thing applies to cancel function, However it does not execute any external calls so it has less possibility of gas issues compare to execute function.

Recommendation

We recommend changing the implementation of _get_current_state function by knowing has_canceled, has_executed & execution_time value through direct storage reads from canceled, executed & execution_time mapping respectively instead of calling _load_actions_set_by_id.

Status

Resolved

Medium Severity Issues

2. Wrong sanity check in constructor leads setting incorrect initial state of contract.

Path

src/bridge_executor/bridge_executor.cairo#L159

Functions

constructor()

Description

In current implementation, assert statement is using the || i.e. or logic operator while it should use && i.e and logic operator otherwise initial value of state variables like minimum_delay, maximum_delay & delay can be set incorrectly which does not follows the assert statement.

Recommendation

We recommend using && operator instead of || in assert statement.

Status

Resolved

Low Severity Issues

3. Missing sanity checks during setting guardian and ethereum_governance_executor address

Path

src/bridge_executor/bridge_executor.cairo#L170

Path

constructor()

Description

In the current implementation of constructor, it is possible to set 0x address for guardian and ethereum_governance_executor. While it is expected that guardian can be set to zero address while setting ethereum_governance_executor to zero address will be drastic and lead the re-deployment of the complete contract.

Recommendation

Add the below assert statement in the constructor() after src/bridge_executor/bridge_executor.cairo#L159.

assert(ethereum_governance_executor.address != ‘’, “Zero address is not allowed”)

Status

Resolved

4. Missing sanity check during update maximum and minimum delay causing current delay value out of bounds.

Path

src/bridge_executor/bridge_executor.cairo#L325

Path

update_minimum_delay() & update_maximum_delay()

Description

In both update_minimum_delay and update_maximum_delay function it is not checked whether the current value of delay is not going outside from the updated minimum_delay or maximum_delay because of that delay value will remain out of maximum or minimum delay bounds value.

If update_minimum_delay or update_maximum_delay function called through governance and if within the same action set there is also update_delay call then it make sense to set those values that makes delay out of bounds however there is no programmatic way to ensure that in the current implementation.

Recommendation

We recommend to add a check in update_maximum_delay and update_minimum_delay function to check that updated value of maximum_delay and minimum_delay doesn't outcast the current set value of delay variable.

In update_minimum_delay

```
assert(self.delay.read() >= minimum_delay,
Errors::MINIMUM_DELAY_TOO_SHORT);
```

In update_minimum_delay

```
assert(self.delay.read() <= maximum_delay,
Errors::MINIMUM_DELAY_TOO_SHORT);
```

Status

Resolved

5. Missing error handling in _execute_transaction keeps non-executable transaction in the smart contract storage

Path

src/bridge_executor/bridge_executor.cairo#L459

Path

_execute_transaction()

Description

In the current implementation, there is no explicit error handling of low-level calls. Instead unwrap_syscall() get call which panics if there is any error and the parent function will revert because of that nature these actions will not attain there right state i.e Failed to execute however that is also missing from the ActionSetState enum. Because of this nature failed execution of actions remain in expired state, However its state should be Failed as it was tried to get executed but failed.

Recommendation

We recommend to introduce a new state i.e Failed in ActionSetState enum and perform error handling in the _execute_transaction() function and update the state if low level calls fails.

Status

Acknowledged

Informational Issues

No Issues Found.

Closing Summary

In this report, we have considered the security of the Lido Starknet Governance Forwarder codebase. We performed our audit according to the procedure described above.

One issue of High severity, one issue of medium severity & three issues of low severity were found. In the End, Lido Starknet Governance Forwarder Team Resolved almost all issues and Acknowledged one informational issue.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Lido Starknet Governance Forwarder smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Lido Starknet Governance Forwarder smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Lido Starknet Governance Forwarder Team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

LIDO STARKNET GOVERNANCE FORWARDER

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com