Zellic

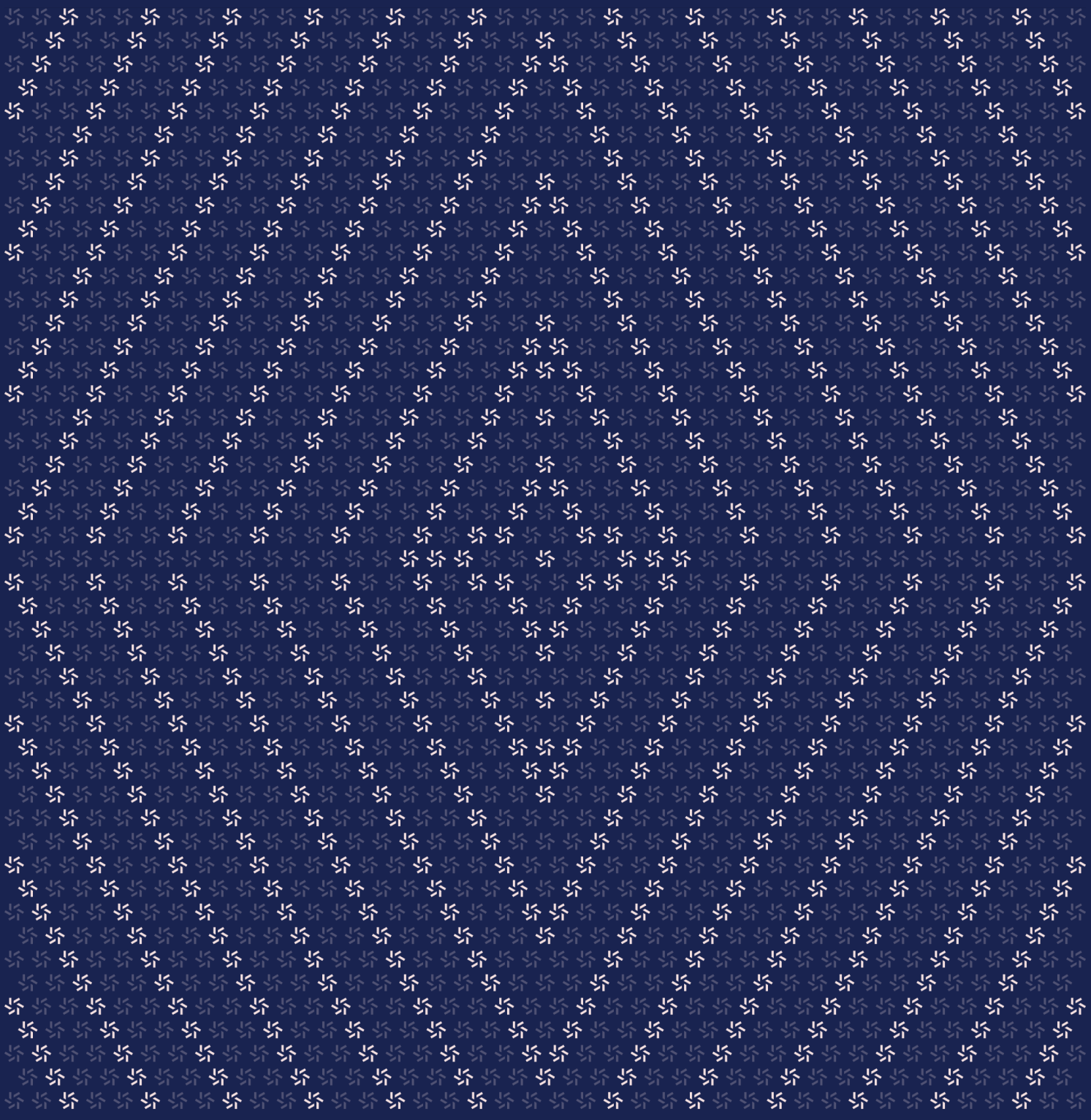
Prepared for
Haichen Shen
Scroll Foundation

Prepared by
Aaron Esau
Vlad Toie
Zellic

January 23, 2024

Lido Gateway

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana, as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Executive Summary

Zellic conducted a security assessment for Scroll Foundation from January 16th to January 19th, 2024. During this engagement, Zellic reviewed Lido Gateway's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is it possible for funds to be permanently locked in the bridge?
- Can attackers drain wstETH locked in the L1 bridge contract?
- Can attackers improperly mint extra wstETH on L2?
- Does the L2 token's supply always match the amount of locked wstETH on L1?

1.2. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- Core Scroll bridge contracts, validators, and so on
- Deployment configuration of in-scope contracts

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.3. Results

During our assessment on the scoped Lido Gateway contracts, we discovered one finding, which was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Scroll Foundation's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	1

2. Introduction

2.1. About Lido Gateway

Scroll is a zkEVM-based ZK rollup on Ethereum that enables native compatibility for existing Ethereum applications and tools. Lido Gateway is used to bridge wstETH between Mainnet and Scroll and relay governance proposals.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Lido Gateway Contracts

Repositories	https://github.com/scroll-tech/scroll ↗ https://github.com/scroll-tech/governance-crosschain-bridges ↗
Versions	Bridge: `69224ebb935d499c055c7859c1c8ade57244249c` Governance: `d023beed13f9b7f77c062b8fee43476788e48343`
Programs	• L1LidoGateway • L2LidoGateway • L2WstETHToken • LidoBridgeableTokens • LidoGatewayManager • ScrollBridgeExecutor
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of eight person-days. The assessment was conducted over the course of four calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Aaron Esau
Engineer
aaron@zellic.io ↗

Vlad Toie
Engineer
vlad@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

January 16, 2024	Start of primary review period
January 19, 2024	End of primary review period

3. Detailed Findings

3.1. L2 token is not necessarily pegged to L1 token

Target	L2LidoGateway		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

Note that it is logically possible for the L2 token's supply to be greater than the L1LidoGateway balance of the L1 token; this is because the L2LidoGateway accepts the token address as a constructor parameter. L2 is ERC-20–implementation agnostic in that it only requires the ability to call `mint` and `burn` on the token address.

Impact

The invariant that L1 locked supply is equal to the L2 supply can possibly be broken if the deployer (e.g., if the code is reused) is not aware of the requirement for L2LidoGateway to be the only minter of the L2 token.

Recommendations

Ensure the only address able to mint L2 tokens is the L2LidoGateway. Alternatively, consider deploying the L2 token from the L2LidoGateway contract and deploying the L2 contracts to Scroll first before the L1 contracts (so that the L2 token's address can be configured on L1).

Remediation

Scroll Foundation noted that they intend to ensure the only minter of L2 tokens is L2LidoGateway:

> We already make sure that L2LidoGateway is the only minter for L2 token. The address of current L2 token is 0xf610a9dfb7c89644979b4a0f27063e9e7d7cda32 ↗, you can see that only the gateway can mint token.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Contract `BridgeExecutorBase` has ability to call `onlyThis` functions

Note that `onlyThis` functions in `BridgeExecutorBase` may be called via `executeTransaction`. This may or may not be intended.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: L1LidoGateway.sol

Function: `_beforeFinalizeWithdrawERC20(address _l1Token, address _l2Token, address None, address None, uint256 None, byte[] None)`

The hook that is called before `finalizeWithdrawERC20` is executed.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Has to be a supported L1 token.
  - **Impact**: The L1 token to be withdrawn.
- `_l2Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Has to be a supported L2 token.
  - **Impact**: The L2 token that has been bridged.

Branches and code coverage (including function calls)

Intended branches

- Ensure that `l1Token` is supported.
  - ☑ Test coverage
- Ensure that `l2Token` is supported.
  - ☑ Test coverage
- Ensure that `withdrawalsEnabled` is true.
  - ☑ Test coverage

Negative behavior

- Should not allow `msg.value > 0`.
  - ☑ Test coverage

Function: `_deposit(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitates the deposit of tokens from L1 to L2.

Inputs

- `_token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Ensured that it is the `l1Token` address.
  - **Impact**: The token to be deposited.
- `_to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The destination address for the deposit.
- `_amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Ensured that it is a nonzero amount.
  - **Impact**: The amount of tokens to be deposited.
- `_data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The gas limit for the deposit.

Branches and code coverage (including function calls)

Intended branches

- Increase the balance of `_token` by `_amount` for `address(this)`.
  - ☑ Test coverage
- Decrease the balance of `_token` by `_amount` for `msg.sender`.
  - ☑ Test coverage
- Generate the message to be sent cross-chain.
  - ☑ Test coverage
- Forward the cross-chain `finalizeDepositERC20` message to L2LidoGateway.
  - ☑ Test coverage
- Ensure that `depositsEnabled` is true.
  - ☑ Test coverage
- Ensure that `token` is supported.

- ☑ Test coverage

Negative behavior

- Should not allow sending tokens if there is not enough balance. Handled in `_transferERC20In`.
  - ☑ Test coverage

5.2. Module: L2LidoGateway.sol

Function: `finalizeDepositERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Finalizes the deposit of ERC-20 tokens from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Ensured that it is supported.
  - **Impact**: The l1Token that has been deposited.
- `_l2Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Ensured that it is supported.
  - **Impact**: The l2Token to be minted.
- `_from`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The depositor of the tokens.
- `_to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The recipient of the tokens.
- `_amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The amount of tokens to be minted.
- `_data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Data to be passed to the callback.

Branches and code coverage (including function calls)

Intended branches

- Perform the callback on `to` with `data`.
  - ☑ Test coverage
- Mint the correct amount of tokens to `to`.
  - ☑ Test coverage
- Ensure that `l1Token` is supported.
  - ☑ Test coverage
- Ensure that `l2Token` is supported.
  - ☑ Test coverage
- Ensure that `depositsEnabled` is true.
  - ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  - ☑ Test coverage
- `msg.value` should not be positive.
  - ☑ Test coverage

Function: `_withdraw(address _l2Token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitates the withdrawing of tokens from L2 to L1.

Inputs

- `_l2Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Ensure it is a supported L2 token.
  - **Impact**: The L2 token to be withdrawn.
- `_to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The address to which the tokens will be sent.
- `_amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Ensured `from` possesses at least `_amount` tokens.
  - **Impact**: The amount of tokens to be withdrawn.
- `_data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.

    - **Impact**: The data to be forwarded cross-chain.
- `_gasLimit`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The gas limit for the cross-chain transaction.

Branches and code coverage (including function calls)

Intended branches

- Ensure that `l2Token` is supported.
    - ☑ Test coverage
- Ensure that withdrawals are enabled.
    - ☑ Test coverage
- Burn the `amount` of tokens from `from`.
    - ☑ Test coverage
- Generate the message to be sent to L1.
    - ☑ Test coverage
- Forward the message to L1.
    - ☑ Test coverage

Negative behavior

- Should not allow sending tokens if there is not enough balance. Handled in `burn`.
    - ☑ Test coverage

5.3. Module: L2WstETHToken.sol

Function: `permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, bytes32 r, bytes32 s)`

Permits `spender` to spend `value` tokens on behalf of `owner` with a valid signature.

Inputs

- `owner`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that the owner is the recovered address from the computed signature (in `isValidSignatureNow`).
    - **Impact**: The owner of the tokens to be approved.
- `spender`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that `spender` is included in the computed signature

(in `isValidSignatureNow`).
    - **Impact**: The spender of the tokens to be approved.
- `value`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that `value` is the approved amount of tokens (in `isValidSignatureNow`).
    - **Impact**: The amount of tokens to be approved.
- `deadline`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that the `deadline` is respected.
    - **Impact**: The deadline of the signature.
- `v`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that `v` is valid in the computed signature (in `isValidSignatureNow`).
    - **Impact**: Component of the ECDSA signature.
- `r`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that `r` is valid in the computed signature (in `isValidSignatureNow`).
    - **Impact**: Component of the ECDSA signature.
- `s`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Ensured that `s` is valid in the computed signature (in `isValidSignatureNow`).
    - **Impact**: Component of the ECDSA signature.

Branches and code coverage (including function calls)

Intended branches

- Ensure that the signature is valid.
    - ☑ Test coverage
- Use a nonce for replay protection.
    - ☑ Test coverage
- Approve the desired amount of tokens.
    - ☑ Test coverage

Negative behavior

- Should not allow using signatures that have been revoked (due to ERC-1271). That is enforced in `isValidSignatureNow`.
    - ☑ Test coverage
- Should not allow the usage of expired signatures.

☑ Test coverage

5.4. Module: LidoGatewayManager.sol

Function: `disableDeposits()`

Function that disables the deposits.

Branches and code coverage (including function calls)

Intended branches

- Set the `isDepositsEnabled` to `false`.
  ☑ Test coverage

Negative behavior

- Should not allow to disable deposits if they are already disabled. Performed in `whenDepositsEnabled` modifier.
  ☑ Test coverage
- Should not allow anyone other than the `depositsDisabler` to disable deposits.
  ☑ Test coverage

Function: `disableWithdrawals()`

Function that disables the withdrawals.

Branches and code coverage (including function calls)

Intended branches

- Set the `isWithdrawalsEnabled` to `false`.
  ☑ Test coverage

Negative behavior

- Should not allow to disable withdrawals if they are already disabled. Performed in `whenWithdrawalsEnabled` modifier.
  ☑ Test coverage
- Should not allow anyone other than the `withdrawalsDisabler` to disable withdrawals.
  ☑ Test coverage

Function: `enableDeposits()`

Function that enables the deposits.

Branches and code coverage (including function calls)

Intended branches

- Set the `isDepositsEnabled` to `true`.
  - ☑ Test coverage

Negative behavior

- Should not allow to enable deposits if they are already enabled.
  - ☑ Test coverage
- Should not allow anyone other than the `depositsEnabler` to enable deposits.
  - ☑ Test coverage

Function: `enableWithdrawals()`

Function that enables the withdrawals.

Branches and code coverage (including function calls)

Intended branches

- Set the `isWithdrawalsEnabled` to `true`.
  - ☑ Test coverage

Negative behavior

- Should not allow to enable withdrawals if they are already enabled.
  - ☑ Test coverage
- Should not allow anyone other than the `withdrawalsEnabler` to enable withdrawals.
  - ☑ Test coverage

Function: `updateDepositsDisabler(address _newDisabler)`

Allows owner to update the address of the deposits disabler.

Inputs

- `_newDisabler`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.

- **Impact**: The address of the new deposits disabler.

Branches and code coverage (including function calls)

Intended branches

- Set the `depositsDisabler` to `_newDisabler`. Currently not tested.
  - ☐ Test coverage

Negative behavior

- Should not allow anyone other than the owner to update the address of the deposits disabler. Not tested.
  - ☐ Test coverage
- Should not allow setting the same address as the current deposits disabler. Not tested and not enforced.
  - ☐ Test coverage
- Should not allow changing the address of the deposits disabler should the deposits be enabled at the moment. Not tested and not enforced.
  - ☐ Test coverage

Function: `updateDepositsEnabler(address _newEnabler)`

Allows owner to update the address of the deposits enabler.

Inputs

- `_newEnabler`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The address of the new deposits enabler.

Branches and code coverage (including function calls)

Intended branches

- Set the `depositsEnabler` to `_newEnabler`. Currently not tested.
  - ☐ Test coverage

Negative behavior

- Should not allow anyone other than the owner to update the address of the deposits enabler. Not tested.
  - ☐ Test coverage
- Should not allow setting the same address as the current deposits enabler. Not tested

and not enforced.
    - ☐ Test coverage
- Should not allow changing the address of the deposits enabler should the deposits be enabled at the moment. Not tested and not enforced.
    - ☐ Test coverage

Function: `updateWithdrawalsDisabler(address _newDisabler)`

Allows owner to update the address of the withdrawals disabler.

Inputs

- `_newDisabler`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The address of the new withdrawals disabler.

Branches and code coverage (including function calls)

Intended branches

- Set the `withdrawalsDisabler` to `_newDisabler`. Currently not tested.
    - ☐ Test coverage

Negative behavior

- Should not allow anyone other than the owner to update the address of the withdrawals disabler. Not tested.
    - ☐ Test coverage
- Should not allow setting the same address as the current withdrawals disabler. Not tested and not enforced.
    - ☐ Test coverage
- Should not allow changing the address of the withdrawals disabler should the withdrawals be disabled at the moment. Not tested and not enforced.
    - ☐ Test coverage

Function: `updateWithdrawalsEnabler(address _newEnabler)`

Allows owner to update the address of the withdrawals enabler.

Inputs

- `_newEnabler`

- **Control**: Fully controlled by the caller.
- **Constraints**: None.
- **Impact**: The address of the new withdrawals enabler.

Branches and code coverage (including function calls)

Intended branches

- Set the `withdrawalsEnabler` to `_newEnabler`. Currently not tested.
  - ☐ Test coverage

Negative behavior

- Should not allow anyone other than the owner to update the address of the withdrawals enabler. Not tested.
  - ☐ Test coverage
- Should not allow setting the same address as the current withdrawals enabler. Not tested and not enforced.
  - ☐ Test coverage
- Should not allow changing the address of the withdrawals enabler should the withdrawals be enabled at the moment. Not tested and not enforced.
  - ☐ Test coverage

5.5. Module: ScrollBridgeExecutor.sol

Function: `constructor(address l2ScrollMessenger, address ethereumGovernanceExecutor, uint256 delay, uint256 gracePeriod, uint256 minimumDelay, uint256 maximumDelay, address guardian)`

Deploys a new L2BridgeExecutor contract with `l2ScrollMessenger`.

Inputs

- `l2ScrollMessenger`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The `l2ScrollMessenger` contract address.
- `ethereumGovernanceExecutor`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The `ethereumGovernanceExecutor` contract address.
- `delay`
  - **Control**: Fully controlled by the caller.

    - **Constraints**: Checked to be greater than `minimumDelay` and less than `maximumDelay`.
    - **Impact**: The delay before which an actions set can be executed.
- `gracePeriod`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Checked to be greater than `MINIMUM_GRACE_PERIOD`.
    - **Impact**: The time period after a delay during which an actions set can be executed.
- `minimumDelay`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Checked to be less than `maximumDelay`.
    - **Impact**: The minimum bound a delay can be set to.
- `maximumDelay`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Checked to be greater than `minimumDelay`.
    - **Impact**: The maximum bound a delay can be set to.
- `guardian`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The guardian address.

Branches and code coverage (including function calls)

Intended branches

- Set all the parameters, including the `L2_SCROLL_MESSENGER` address.
    - ☑ Test coverage

6. Assessment Results

At the time of our assessment, the reviewed code was partly deployed to the Ethereum Mainnet and Scroll.

During our assessment on the scoped Lido Gateway contracts, we discovered one finding, which was informational in nature. Scroll Foundation acknowledged the finding and implemented a fix.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.