LIDO STETH ON OPTIMISM SECURITY AUDIT REPORT

Oct 14, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Lido stETH on the Optimism protocol allows users to bridge not only non-rebasable wstETH tokens from Ethereum to Optimism-like rollups but also to transfer stETH from L1 to L2 via the bridge. Most importantly, stETH on L2 will continue to receive rebases, which means users will continue receiving rewards in the form of stETH balance rebases.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Lido
Project name	stETH on Optimism
Timeline	April 23 2024 - June 21 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
23.04.2024	3f2ef873244b83971208efb95da3cf672f16f4d8	Commit for the audit
24.04.2024	792071cdeaf61de927cc144e8c1c02d5f5996a01	Commit with updates
10.06.2024	a31049ac8828d6d6a214b63279ff678101d55308	Commit for the reaudit
21.06.2024	8f19e1101a211c8f3d42af7ffcb87ab0ebcf750c	Commit with updates

Project Scope

The audit covered the following files:

File name	Link
contracts/BridgingManager.sol	BridgingManager.sol
contracts/lib/DepositDataCodec.sol	DepositDataCodec.sol

File name	Link
contracts/lib/ECDSA.sol	ECDSA.sol
contracts/lib/SignatureChecker.sol	SignatureChecker.sol
contracts/lib/UnstructuredRefStorage.sol	UnstructuredRefStorage.sol
contracts/lib/UnstructuredStorage.sol	UnstructuredStorage.sol
contracts/lido/TokenRateNotifier.sol	TokenRateNotifier.sol
contracts/token/ERC20Bridged.sol	ERC20Bridged.sol
contracts/token/ERC20BridgedPermit.sol	ERC20BridgedPermit.sol
contracts/token/ERC20Core.sol	ERC20Core.sol
contracts/token/PermitExtension.sol	PermitExtension.sol
contracts/token/ERC20RebasableBridgedPermit.sol	ERC20RebasableBridgedPermit.sol
contracts/token/ERC20Metadata.sol	ERC20Metadata.sol
contracts/token/ERC20RebasableBridged.sol	ERC20RebasableBridged.sol
contracts/optimism/L1ERC20ExtendedTokensBridge.sol	L1ERC20ExtendedTokensBridge.sol
contracts/optimism/L1LidoTokensBridge.sol	L1LidoTokensBridge.sol
contracts/optimism/TokenRateOracle.sol	TokenRateOracle.sol
contracts/optimism/OpStackTokenRatePusher.sol	OpStackTokenRatePusher.sol

File name	Link
contracts/optimism/L2ERC20ExtendedTokensBridge.sol	L2ERC20ExtendedTokensBridge.sol
contracts/optimism/RebasableAndNonRebasableTokens.sol	RebasableAndNonRebasableTokens.sol
contracts/optimism/CrossDomainEnabled.sol	CrossDomainEnabled.sol
contracts/utils/Versioned.sol	Versioned.sol
contracts/optimism/TokenRateAndUpdateTimestampProvider.sol	TokenRateAndUpdateTimestampProvider.sol

Deployments

Ethereum:mainnet

File name	Contract deployed on mainnet	Comment
TokenRateNotifier.sol	0xe6793B9e4FbA7DE0ee833F9D02bba7DB5EB27823	
OpStackTokenRatePusher.sol	0xd54c1c6413caac3477AC14b2a80D5398E3c32FfE	
L1LidoTokensBridge.sol	0x168Cfea1Ad879d7032B3936eF3b0E90790b6B6D4	

Optimism:mainnet

File name	Contract deployed on mainnet	Comment
ERC20BridgedPermit.sol	0xFe57042De76c8D6B1DF0E9E2047329fd3e2B7334	
OssifiableProxy.sol	0x76A50b8c7349cCDDb7578c6627e79b5d99D24138	Proxy for the ERC20RebasableBridgedPermit

File name	Contract deployed on mainnet	Comment
ERC20RebasableBridgedPermit.sol	0xe9b65dA5DcBe92f1b397991C464FF568Dc98D761	
OssifiableProxy.sol	0x294ED1f214F4e0ecAE31C3Eae4F04EBB3b36C9d0	Proxy for the TokenRate Oracle
TokenRateOracle.sol	0x4bF0d419793d8722b8391efaD4c9cE78F460CEd3	
L2ERC20ExtendedTokensBridge.sol	0x2734602C0CEbbA68662552CacD5553370B283E2E	

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	1
Medium	1
Low	18

ID	Name	Severity	Status
H-1	stETH liquidity problem	High	Fixed
M-1	Rounding errors	Medium	Fixed
L-1	`name` and `symbol` are not checked	Low	Fixed
L-2	Unrestricted `initialize`	Low	Fixed
L-3	Tx with an incorrect rate will not revert	Low	Fixed
L-4	CEXes should work with shares	Low	Fixed
L-5	The rate can differ inside one block	Low	Fixed
L-6	`maxAllowedTokenRateDeviationPerDay_` is not checked	Low	Fixed
L-7	`tokenRate_` and `rateL1Timestamp_` are not checked	Low	Fixed
L-8	`MAX_ALLOWED_L2_TO_L1_CLOCK_LAG` should be updatable	Low	Acknowledged

L-9	Optimizations	Low	Fixed
L-10	The rate can be set to 0	Low	Acknowledged
L-11	Important notes	Low	Fixed
L-12	A strange comment	Low	Fixed
L-13	Transfers of zero value	Low	Acknowledged
L-14	The `onlySupportedL1L2TokensPair` modifier and the `_getL1Token` function can be unified	Low	Acknowledged
L-15	Missing zero address checks	Low	Fixed
L-16	`PermitExtension.eip712Domain()` may return incorrect values for `name` and `version`	Low	Acknowledged
L-17	Withdrawn stETH gets stuck on the bridge if the recipient is an stETH address on L1	Low	Fixed
L-18	`roundedUpNumberOfDays` in `_isTokenRateWithinAllowedRange()` can be calculated more precisely	Low	Fixed

1.6 Conclusion

During the audit, we checked that:

1. It is **impossible to manipulate** `rate` values via `data` parameter adjustments when they are encoded/decoded in the DepositDataCodec contract:
   The `data` parameter represents bytes which can be passed inside cross domain messages. The `DepositData` struct is constructed in a way that there are no possible intersections between `bytes data`, `rate` and `timestamp`.

2. DepositDataCodec **correctly works** with bytes arrays:
   That contract encodes the `DepositData` struct using bytes concatenation and then decodes it using offsets, which accurately represent the size of each decoded field.

3. It is empirically **impossible to manipulate** different stETH to wstETH rates on L1 and L2s. If there is a chance to manipulate the difference in rates, then it is also possible to manipulate it just on L1 (sandwiching of the token rate update):
   If stETH is being bridged, then it is initially wrapped to wstETH and then minted on the other side (if it is a transfer from L1 to L2 then wstETH is first minted on the L2 side an then unwrapped to stETH). All the cross chain messages from L1 to L2 go together with a stETH to wstETH rate which is then pushed to the rate oracle on the L2 side. While there may be potential issues such as slashings and rebases, these risks also exist without bridging functionality and can be viewed as arbitrage opportunities. Overall, everything mentioned doesn't lead to any abuses which could make the system exploitable.

4. Contracts implementation cannot be used in any way except through a **proxy**:
   All contracts under proxies use the `Versioned` contract which helps to manage contract versions. It sets and checks the `CONTRACT_VERSION_POSITION` storage variable in the implementation contracts to prevent their initialization.

5. The **upgrade will not break** the current implementation of L1Bridge, L2Bridge, wstEth on L2:
   There are no potential storage clashings as there are no new storage variables introduced in unexpected places. `UnstructuredStorage` and `UnstructuredRefStorage` libraries are used to store and access storage variables.

6. **DoS** attacks with dummy updates **are impossible**:
   Only special admin roles have the authority to pause deposits or withdrawals, which can be used in case of an emergency. In other scenarios, admin roles don't allow to modify the protocol paramteres, ensuring the protocol remains functional.

7. **BASIS_POINT_SCALE = 1e4** is **sufficient** for accounting for daily increases/decreases in rewards:
   `BASIS_POINT_SCALE` is used to calculate upper and lower bounds for the token rate deviation. It scales the previously reported token rate and compares it with the current reported one. The

calculation is implemented correctly as the `BASIS_POINT_SCALE` is used in both the numerator and denominator to derive a defined percentage from the stored token rate.

8. All the **ERC20 functions** are **implemented correctly**:
   There are`ERC20Bridged` and `ERC20RebasableBridged` which implement basic ERC20 functions such as `transfer`, `transferFrom` and `approve`. There are also additional functions such as `bridgeMint`, `bridgeBurn`, `bridgeMintShares` and `bridgeBurnShares` which allow to mint or burn tokens or shares. All those functions are implemented in a secure way.

9. **EIP5267** is correctly implemented:
   The `PermitExtension` contract implements EIP712 and ERC2612 interaces alongside EIP5267 domain metadata. It introduces a special slot and functions to store and manage domain separator data.

10. **Permit call** to the wstEth on L2 with stETH on L2 specified as the owner (it will fail):
    If there would be an attempt to call the `permit` function with the `stETH` on L2 specified as the `owner`, the transaction would revert due to `stETH` returning an empty string on an `isValidSignature` callback.

11. All the functions that require **restricted access** are properly secured:
    There are special roles in the `BridgingManager` contract that enable or disable deposits and withdrawals, and the `TokenRateNotifier` inherits from ownable to make observers managed by the special contract owner.

12. **Pausing the bridge** for a long time won't affect the system:
    If there is an emergency, then deposits and withdrawals can be paused by an account holding a special role. It won't affect the token rate as it will still be pushed via an observer and accepted on the L2 side by the `TokenRateOracle` contract.

13. **Possible shares rounding attacks** during wrapping-unwrapping on L1 and L2:
    During the bridging process, stETH is wrapped to wstETH, which is done at the current rate on the L1 side. When it is bridged, the same amount of wstETH is minted on the L2 and then unwrapped to the stETH. All the process is done by multiplying the transferred amount of wstETH by the conversion rate together with the rate decimals.

14. **Bridging** of the **small amount** of tokens:
    It is possible to initiate bridging with a small amount of tokens. If stETH is bridged, it gets wrapped to the wstETH at the current rate, the exact same amount of wstETH is minted on the L2 and then unwrapped to stETH at the current rate on the L2. If wstETH is bridged, it is minted on the L2 side without wrapping. It is also possible to initiate bridging with zero tokens, which will lead to empty `Transfer` events being emitted. Still, it won't have any bad effect on the whole system as nothing won't be minted.

15. `TokenRateOracle` can be **paused and resumed** securely:
    There is a functionality that allows `TokenRateOracle` to be set on pause by the address granted a special role. It can be resumed together with providing the new token rate, which will be used as the actual one. All necessary checks ensure that there is a correct new token rate and rate update timestamp.

Besides conducting a security audit, the same team of auditors completed deployment verification. The following aspects were thoroughly checked:

1. Deployed contracts' bytecode matches the audited contracts' source code.
2. All the deployed contracts are initialized correctly following the upgrade specification.
3. It is impossible to initialize any of the used implementations for proxies due to the use of the `Versioned` contract, which sets the contract version to `type(uint256).max`.
4. Contracts utilize the correct Optimism messenger contracts.
5. `L1LidoTokensBridge` points to the proxy on the Optimism network side, which uses the `L2ERC20TokenBridge` implementation, which would be upgraded to the `L2ERC20ExtendedTokensBridge`.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	stETH liquidity problem
Severity	High
Status	Fixed in a31049ac

Description

L2ERC20Bridge mints stETH without minting a corresponding amount of wstETH on L2 L2ERC20ExtendedTokensBridge.sol#L172. This can lead to insolvency issues, affecting users who have bridged wstETH on L2 and wrapped them to stETH. There is a chance (this can be forced by a malicious user without any losses) that the stETH contract on L2 might lack wstETH in its balance. This will require users who wrapped wstETH to stETH on L2 to:

- transfer stETH from L2 to L1;
- wrap stETH to wstETH on L1;
- transfer wstETH from L1 to L2.

Recommendation

We recommend minting wstETH on L2, locking them on the stETH contract, and subsequently transferring stETH to the user on L2.

Client's commentary

> Fixed

2.3 Medium

M-1	Rounding errors
Severity	Medium
Status	Fixed in a31049ac

Description

Due to rounding errors users will always retain some dust on L2 ERC20RebasableBridged.sol#L86-L94. This problem can be mitigated by adding a method that allows users to unwrap stETH shares into wstETH shares.

Recommendation

We recommend adding a method that allows users to unwrap stETH shares into wstETH shares.

Client's commentary

> Fixed: unwrapShares() method was added.

2.4 Low

L-1	`name` and `symbol` are not checked
Severity	Low
Status	Fixed in a31049ac

Description

`name_` and `symbol_` are not checked, but if they both have zero length then contract still remain uninitialized ERC20Bridged.sol#L59-L62.

Recommendation

We recommend adding a check that either `name_` or `symbol_` has a non-zero length.

Client's commentary

> Fixed

L-2	Unrestricted `initialize`
Severity	Low
Status	Fixed in a31049ac

Description

`initialize` can be called on the proxy after the upgrade ERC20BridgedPermit.sol#L35-L38.

Recommendation

We recommend adding a check that `initialize` cannot be called on the previously initialized version, as it is designed for L1ERC20Bridge and L2ERC20Bridge.

Client's commentary

> Fixed

L-3	Tx with an incorrect rate will not revert
Severity	Low
Status	Fixed in a31049ac

Description

Rate updates should always be checked to be in the allowed range TokenRateOracle.sol#L126-L128
Currently an incorrect rate can be emitted from here TokenRateOracle.sol#L121.

Recommendation

We recommend first checking that the rate is within the correct range before verifying that it was updated in the same block.

Client's commentary

> The event was fixed

L-4	CEXes should work with shares
Severity	Low
Status	Fixed in a31049ac

Description

CEXes should fetch the token rate from the specific chain for deposits/withdrawals TokenRateOracle.sol#L121. If a CEX uses the token rate from L1 for L2 deposits/withdrawals, it could result in bad debt for the exchange.

Recommendation

We recommend adding this information to the documentation.

Client's commentary

> fixed: Comment in TokenRateOracle was updated

L-5	The rate can differ inside one block
Severity	Low
Status	Fixed in a31049ac

Description

A malicious user can front-run a rate update with another rate update, resulting in two token rate updates with different token rates for the same block TokenRateOracle.sol#L120-L123. This will lead to a postponed rate update on L2.

Recommendation

We recommend making a rate update more centralized and using only `TokenRateNotifier` for it. Additionally, we recommend adding a mapping to the `TokenRateNotifier` which will account for the number of tx in the block. This number can be used in the oracle to order tx and correctly update rates.

Client's commentary

> Fixed by sending L1 time calculated using core protocol ref slot.

L-6	`maxAllowedTokenRateDeviationPerDay_` is not checked
Severity	Low
Status	Fixed in a31049ac

Description

`maxAllowedTokenRateDeviationPerDay_` cannot exceed the value of `BASIS_POINT_SCALE` TokenRateOracle.sol#L74.

Recommendation

We recommend adding a check that `maxAllowedTokenRateDeviationPerDay_` is less than `BASIS_POINT_SCALE`.

Client's commentary

> Fixed: Check was added

L-7	`tokenRate_` and `rateL1Timestamp_` are not checked
Severity	Low
Status	Fixed in a31049ac

Description

`TokenRateOracle` can be initialized with random data TokenRateOracle.sol#L79.

Recommendation

We recommend adding a check that `tokenRate_` is in the range [1*10**18 ; 2*10**18] and `rateL1Timestamp_` >= current L2 timestamp.

Client's commentary

> Fixed by adding additional checks for rate and time.

L-8	`MAX_ALLOWED_L2_TO_L1_CLOCK_LAG` should be updatable
Severity	Low
Status	Acknowledged

Description

`MAX_ALLOWED_L2_TO_L1_CLOCK_LAG` should be updatable for some extreme cases TokenRateOracle.sol#L115.

Recommendation

We recommend making `MAX_ALLOWED_L2_TO_L1_CLOCK_LAG` updatable.

Client's commentary

> Acknowledged. MAX_ALLOWED_L2_TO_L1_CLOCK_LAG can be changed by updating the contract.

L-9	Optimizations
Severity	Low
Status	Fixed in a31049ac

Description

Token rate range checks can be skipped if the token rate has the same value as the previous one TokenRateOracle.sol#L146-L162.
During the removal of an observer, the removed one is overwritten by the last one: TokenRateNotifier.sol#L79. It can be optimized by doing so only if the removed one is not the last one.
`TOKEN_RATE_ORACLE.decimals()` is called every time when `ERC20RebasableBridged._getTokenRateAndDecimal()` is called: ERC20RebasableBridged.sol#L291.

Recommendation

We recommend adding a check that the token rate doesn't change and, in this case, skip range checks. Also, we recommend modifying the condition from `if (observers.length > 1)` to `if (observerIndexToRemove != observers.length - 1)`. Additionally, we recommend setting the oracle's decimals as an immutable in `ERC20RebasableBridged` or keeping it in the proxy's storage with the addition of a corresponding setter function.

Client's commentary

> Fixed

L-10	The rate can be set to 0
Severity	Low
Status	Acknowledged

Description

If the rate is not updated for some period of time, it can be set to 0 TokenRateOracle.sol#L155-L158.

Recommendation

We recommend adding a check that requires admin actions if the token rate hasn't been updated for an extended period of time.

Client's commentary

> Acknowledged, but we've implemented a new method to pause and resume token rate updates.

L-11	Important notes
Severity	Low
Status	Fixed in a31049ac

Description

Change `//` to `/` :Versioned.sol#L6
`_checkContractVersion` is not used and can be removed Versioned.sol#L39-L44
`_updateContractVersion` is not used and can be removed Versioned.sol#L53-L56
There is an important misspelling, it is better to add a comment that without this misspelling the contract storage will be broken ERC20Metadata.sol#L33
`eip712Domain()` can be external PermitExtension.sol#L89
`_isCallerBridgeOrMessegerWithTokenRatePusher`: there is a misspelled word `Messeger` in the function name TokenRateOracle.sol#L164
`ErrorUnsupportedL1Token` is not used and can be removed RebasableAndNonRebasableTokens.sol#L74
`whenDepositsEnabled` doesn't accept any parameters, so it can be used without parentheses L2ERC20ExtendedTokensBridge.sol#L122
`_encodeInputDepositData` doesn't have a comment L1ERC20ExtendedTokensBridge.sol#L167
There are missing spaces at the lines (between if operator and opening parenthesis): L1ERC20ExtendedTokensBridge.sol#L159; L1LidoTokensBridge.sol#L53; L2ERC20ExtendedTokensBridge.sol#L68; L2ERC20ExtendedTokensBridge.sol#L171; L2ERC20ExtendedTokensBridge.sol#L190; TokenRateOracle.sol#L155; TokenRateOracle.sol#L165; TokenRateOracle.sol#L168; TokenRateOracle.sol#L200
Misplaced spaces (before commas) at the lines: ERC20RebasableBridged.sol#L238; ERC20RebasableBridged.sol#L319; ERC20RebasableBridged.sol#L334
`decimals` variables are shadowing ERC20 `decimals` ERC20RebasableBridged.sol#L281; ERC20RebasableBridged.sol#L286

Recommendation

We recommend applying the recommendations for each problem.

Client's commentary

> Fixed

L-12	A strange comment
Severity	Low
Status	Fixed in a31049ac

Description

`isInitialized` is still used to determine whether the admin has been initialized or not BridgingManager.sol#L17.

Recommendation

We recommend updating the comment.

Client's commentary

> Fixed

L-13	Transfers of zero value
Severity	Low
Status	Acknowledged

Description

Due to rounding errors, non-zero stETH amount can be converted to 0 wstETH:
L1ERC20ExtendedTokensBridge.sol#L137
L2ERC20ExtendedTokensBridge.sol#L152
ERC20RebasableBridged.sol#L236

So, no tokens will be actually transfered, but a `Transfer` event with a non-zero amount will be emitted:
ERC20RebasableBridged.sol#L362,

and non-zero `allowance` will be spent: ERC20RebasableBridged.sol#L161.

It can be done even if the caller or `from_` address has a zero balance. If some offchain service relies on `Transfer` events, it could lead to broken invariants.

The original stETH on L1 follows the same logic: StETH.sol#L375.

Recommendation

We recommend restricting the transfer of 0 shares because it is an abnormal behavior of the system for a user to spend a non-zero approval to transfer 0 amount of tokens.

Client's commentary

> Acknowledged. zero transfers are legit ones for ERC20

L-14	The `onlySupportedL1L2TokensPair` modifier and the `_getL1Token` function can be unified
Severity	Low
Status	Acknowledged

Description

There is the `onlySupportedL1L2TokensPair` modifier defined at the line RebasableAndNonRebasableTokens.sol#L51. It is used to check whether the user provided a correct pair of token addresses in the `L1ERC20ExtendedTokensBridge` contract. In the `L2ERC20ExtendedTokensBridge` contract, a different approach is employed where `_getL1Token` is used to get the corresponding token address on the L1 side. The same method can be adopted in the `L1ERC20ExtendedTokensBridge` so that users won't need to provide two token addresses.

Recommendation

We recommend implementing the `_getL2Token` function in the `L1ERC20ExtendedTokensBridge` contract which will return the corresponding token address on L2.

Client's commentary

> Acknowledged. L1Bridge is required to provide one token in deposit/depositTo methods and L2Bridge only one token in withdraw/withdrawTo methods. Thus, unifying doesn't make sense in my opinion.

L-15	Missing zero address checks
Severity	Low
Status	Fixed in a31049ac

Description

There are two constructors defined at the lines L1ERC20ExtendedTokensBridge.sol#L37 and L2ERC20ExtendedTokensBridge.sol#L43. They accept parameters `l2TokenBridge_` and `l1TokenBridge_` respectively but don't check these values for zero addresses.

Recommendation

We recommend adding checks that `l2TokenBridge_` and `l1TokenBridge_` are not equal to zero addresses.

Client's commentary

> Fixed

L-16	`PermitExtension.eip712Domain()` may return incorrect values for `name` and `version`
Severity	Low
Status	Acknowledged

Description

`name` and `version` for the domain separator in `PermitExtension` are stored in two locations.
The first place is in the bytecode, set by the constructor of `PermitExtension's` base class `EIP712`:
draft-EIP712.sol#L35-L36.

The second place is in the storage, set by `PermitExtension._initializeEIP5267Metadata()` during the contract's initialization:
PermitExtension.sol#L17
PermitExtension.sol#L116.

The bytecode version is used to build the domain separator:
draft-EIP712.sol#L70.

The storage version is returned by `PermitExtension.eip712Domain()`
PermitExtension.sol#L88.

There are no checks during the initialization of `ERC20BridgedPermit's` and `ERC20RebasableBridgedPermit's` that these pairs are the same:
ERC20BridgedPermit.sol#L35
ERC20BridgedPermit.sol#L41
ERC20RebasableBridgedPermit.sol#L39.

So, it can lead to a situation where `PermitExtension.eip712Domain()` returns an incorrect `name` and `version`.

Recommendation

We recommend using `EIP712Upgradeable` from OpenZeppelin's `contracts-upgradeable` as the base contract for `PermitExtension` instead of `EIP712` from OpenZeppelin's `contracts`.

Client's commentary

> Client: Acknowledged since using EIP712Upgradeable requires to use another OZ version which we would like to avoid. Mitigates by checking during deployment and testing.

> MixBytes: There is also an option to add a check to `PermitExtension._initializeEIP5267Metadata()` that `keccak256(bytes(name_)) == _HASHED_NAME` and `keccak256(bytes(version_)) == _HASHED_VERSION`

L-17	Withdrawn stETH gets stuck on the bridge if the recipient is an stETH address on L1
Severity	Low
Status	Fixed in a31049ac

Description

stETH on L1 doesn't permit transfers to the token's own address:
StETH.sol#L444.
However, there are no restrictions in `L2ERC20ExtendedTokensBridge.withdrawTo()` to use such address as a recipient:
L2ERC20ExtendedTokensBridge.sol#L97
So, if someone uses it as a withdrawal address, tokens will be successfully burnt on L2 but the finalization on L1 will always revert:
L1ERC20ExtendedTokensBridge.sol#L114.

Sending tokens to any incorrect address will likely lead to the loss of funds. At the same time having tokens stuck on the bridge is also an undesirable scenario.

Recommendation

We recommend adding a check in `L2ERC20ExtendedTokensBridge.withdrawTo()` that the recipient is not an stETH address on L1.

Client's commentary

> Fixed by checking address.

L-18	`roundedUpNumberOfDays` in `_isTokenRateWithinAllowedRange()` can be calculated more precisely
Severity	Low
Status	Fixed in a31049ac

Description

If `rateL1TimestampDiff` is divided by `ONE_DAY_SECONDS` without remainder, `roundedUpNumberOfDays` will be 1 more than it should be:
TokenRateOracle.sol#L150

Recommendation

We recommend changing

```
uint256 roundedUpNumberOfDays =
    rateL1TimestampDiff / ONE_DAY_SECONDS + 1;
```

to

```
uint256 roundedUpNumberOfDays =
    (rateL1TimestampDiff + ONE_DAY_SECONDS - 1) / ONE_DAY_SECONDS;
```

Client's commentary

> Fixed

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes