Lido

stETH on Optimism

by Ackee Blockchain

18.06.2024

Contents

1. Document Revisions

0.1	Draft report	20.05.2024
1.0	Final report	20.05.2024
1.1	Fix review	05.06.2024
1.2	Fix review	07.06.2024
1.3	Review of the extended scope	18.06.2024

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Wake is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member's Name	Position
Andrey Babushkin	Lead Auditor
Michal Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Revision 1.0

Lido engaged Ackee Blockchain to perform a security review of the Lido protocol with a total time donation of 15 engineering days in a period between May 6 and May 17, 2024, with Andrey Babushkin as the lead auditor.

The audit was performed on the commit 9d6f66c [1] and the scope was the following:

- contracts/lido/TokenRateNotifier.sol
- contracts/optimism/CrossDomainEnabled.sol
- contracts/optimism/L1ERC20ExtendedTokensBridge.sol
- contracts/optimism/L1LidoTokensBridge.sol
- contracts/optimism/L2ERC20ExtendedTokensBridge.sol
- contracts/optimism/OpStackTokenRatePusher.sol
- contracts/optimism/RebasableAndNonRebasableTokens.sol
- contracts/optimism/TokenRateOracle.sol
- contracts/token/ERC20Bridged.sol
- contracts/token/ERC20BridgedPermit.sol
- contracts/token/ERC20Core.sol
- contracts/token/ERC20Metadata.sol
- contracts/token/ERC20RebasableBridged.sol
- contracts/token/ERC20RebasableBridgedPermit.sol
- contracts/token/PermitExtension.sol

We used Wake testing framework for cross-chain fuzzing of the protocol. This yielded the [L1] and [W2] findings. Additionally, the fuzzing campaign was used to confirm the [L2] finding detected using [Wake] as a static analysis tool. A static analysis detector also detected the [I3] finding. As a part of the audit, we also performed upgradeability testing to ensure a hassle-free upgrade process. The upgradeability testing did not yield any findings.

We also conducted a thorough manual review of the codebase and took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- ensuring access controls are not too relaxed or too strict,
- validating the integration into the Optimism stack,
- making sure the cross-chain architecture and operations are properly secured,
- ensuring the deposits to and withdrawals from L2 cannot lead to double spending,
- making sure the token rate cannot be manipulated,
- ensuring the arithmetic of the system is correct,
- looking for common issues such as data validation.

Our review resulted in 15 findings, ranging from Info to Low severity. The most severe one is [L1] mentioned above.

Ackee Blockchain recommends Lido:

- validate the arithmetic of the system to limit rounding errors,
- make sure permits are prepared for smart accounts,
- implement proper data validation,
- fix minor problems with the documentation, following best practices and the overall code quality,

- address all other reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The review was done on the given commit: `a479315` [2].

See Revision 1.1 for the review of the updated codebase and additional information we consider essential for the current scope.

Most of the issues were addressed. Issues W1, W2, W6, W7, and I5 were acknowledged, issues W8 and I4 were partially fixed.

The contracts that work with the token rate were updated to reflect the new precision of the token rate. A new contract `contracts/optimism/TokenRateAndUpdateTimestampProvider.sol` was added, and the `contracts/optimism/TokenRateOracle.sol` contract was largely refactored. In addition to using 27 decimals for the token rate, the token rate updates can now be paused on L2, and multiple additional validations for the token rate were added.

The Ackee Blockchain team reviewed the fixes and confirmed that the issues were addressed. However, the newly added code was not reviewed, and the review was only focused on the changes that addressed the reported issues.

Revision 1.2

The review was done on the given commit: `a31049a` [3].

The codebase was updated to completely fix issues W8 and I4. The Ackee Blockchain team reviewed the fixes and confirmed that the issues were addressed. However, the newly added code in Revision 1.1 was not reviewed, and the review was only focused on the changes that addressed the reported issues.

Revision 1.3

The review was performed on the given commit: `8f19e11` [4].

Lido engaged Ackee Blockchain to perform a security review of the Lido protocol with a total time donation of 1.5 engineering days in a period between June 17 and June 18, 2024, with Andrey Babushkin as the lead auditor. The scope of the audit was extended to include all the contracts in the repository and all the changes that were not reviewed in the previous revisions. The scope was extended to include the following contracts and all the changes that were not reviewed in the previous revisions:

- contracts/lib/DepositDataCodec.sol
- contracts/lib/UnstructuredRefStorage.sol
- contracts/lib/UnstructuredStorage.sol
- contracts/optimism/TokenRateAndUpdateTimestampProvider.sol
- contracts/optimism/TokenRateOracle.sol
- contracts/proxy/OssifiableProxy.sol
- contracts/utils/Versioned.sol
- contracts/BridgingManager.sol

During the review, we paid special attention to the changes made in the contracts `contracts/optimism/TokenRateAndUpdateTimestampProvider.sol` and `contracts/optimism/TokenRateOracle.sol`. A detailed overview of the changes can be found in the Revision 1.3.

The review resulted in unused code occurrences that were fixed at the time of the review, and the fixes are already included in the commit `8f19e11`. Other than that, no new findings were detected.

[1] full commit hash: 9d6f66c085c03652345df1c4c948ef45e39db42b

[2] full commit hash: a479315aef0197ccdfa9b87e0eae4eb9e53e3950

[3] full commit hash: a31049ac8828d6d6a214b63279ff678101d55308

[4] full commit hash: 8f19e1101a211c8f3d42af7ffcb87ab0ebcf750c

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
L1: Insufficient token rate precision	Low	1.0	Fixed
L2: `unwrap` inconsistent tokens amount in event	Low	1.0	Fixed
W1: Usage of `solc` optimizer	Warning	1.0	Acknowledged
W2: ERC-20 `transferFrom` emits `Approval`	Warning	1.0	Acknowledged
W3: False comments	Warning	1.0	Fixed
W4: Limited ERC-2612 use-case with ERC-1271	Warning	1.0	Fixed
W5: Use of a deprecated function	Warning	1.0	Fixed

	Severity	Reported	Status
W6: Initializers can be front-run	Warning	1.0	Acknowledged
W7: Linear calculation of the allowed token rate deviation	Warning	1.0	Acknowledged
W8: Insufficient data validation	Warning	1.0	Fixed
I1: Uncached `.length` in for loop	Info	1.0	Fixed
I2: Inconsistent modifiers order	Info	1.0	Fixed
I3: Unused code	Info	1.0	Fixed
I4: Typos	Info	1.0	Fixed
I5: `_mintShares` can return `tokensAmount` to save gas	Info	1.0	Acknowledged

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

The stETH on Optimism introduces the staked ETH and the wrapped staked ETH tokens on the Optimism blockchain. The system is composed of multiple contracts, which include the ERC20 token contracts, bridges, and the oracle. Users can send stETH or wstETH tokens to the bridge contract on L1, which will lock the tokens and mint the corresponding tokens on the Optimism blockchain. The oracle contract is deployed on L2 and provides the correct rate between the internal token representation, shares, and the stETH tokens.

On L1, stETH tokens are rebasable, i.e., balances are represented by shares owned by users. The rate between shares and stETH tokens is computed by the ratio of the total ETH locked in the stETH contract and the total shares minted. The wstETH contract wraps stETH tokens and one wstETH equals one share in the internal stETH representation of the balance.

On L2, wstETH and stETH tokens are organized similarly, except that stETH is a wrapper around wstETH tokens. When one transfers stETH tokens from L1 to L2, the bridge contract on L1 first wraps the stETH tokens into wstETH tokens and locks wstETH. The bridge contract sends a message to L2 with the number of wstETH tokens locked along with the current token rate from the stETH contract. The bridge contract on L2 receives the message and mints the corresponding amount of wstETH tokens on L2. Also, the bridge pushes the new rate to the oracle contract. Then, the wstETH are wrapped into stETH tokens on L2. The stETH tokens are internally represented by shares, and the actual number of stETH tokens is computed dynamically based on the rate from the oracle. This double-wrapping mechanism allows having rebasable tokens on L2 computed using the same rate as on L1.

Users can deposit either stETH or wstETH tokens to Optimism. If the user chooses to send wstETH, the process is the same as described above, except that the bridge contract on L2 does not wrap the wstETH tokens into stETH tokens, and the transfer is done directly.

The withdrawal process from L2 to L1 is similar to the deposit process but in reverse. The user sends stETH tokens to the bridge contract on L2, which unwraps the stETH tokens into wstETH by burning the amount of stETH shares computed from the token amount and the current rate from the oracle. Next, the bridge contract burns the wstETH tokens and sends a message to L1 with the number of wstETH shares burned. The bridge contract on L1 receives the message and unlocks the corresponding amount of wstETH tokens on L1. Finally, these wstETH tokens are unwrapped to stETH tokens on L1. The withdrawal process for wstETH tokens is similar, except that the bridge contract on L2 does not unwrap the wstETH tokens into stETH tokens.

The system relies on the oracle contract to provide the correct rate between the internal representation of the tokens and the actual stETH tokens. The rate is updated with every deposit from L1 to L2 and also it can be updated manually by anyone permissionlessly.

Contracts

Contracts we find important for better understanding are described in the following section.

TokenRateNotifier

The contract is used as a notifier of the rebase event for the registered observers. Observers are added and removed from the contract by the owner.

The `handlePostTokenRebase` function can be called by anyone and it will notify all the registered observers that the token rate should be updated.

OpStackTokenRatePusher

The contract is deployed on L1 and used to push the token rate from L1 to L2 by calling the `pushTokenRate` function. This contract is registered as one of the observers in the `TokenRateNotifier` contract. It retrieves the current rate from the stETH contract and sends it to the `TokenRateOracle` contract on L2 through the Optimism messenger.

TokenRateOracle

The contract is deployed on L2 and used to store the current token rate. The contract is the only source of the token rate for the stETH contract on L2. The rate can be updated either directly through the Optimism messaging from the `OpStackTokenRatePusher` contract on L1, or by `L2ERC20ExtendedTokensBridge` when the bridge processes a deposit from L1.

L1LidoTokensBridge

The contract is a part of the bridging mechanism between L1 and L2. It is used to lock and unlock tokens on L1. When a user sends stETH tokens to the contract, it wraps the stETH tokens into wstETH tokens and locks them. Next, it composes a message with the current token rate and the number of the locked tokens and sends it to L2. The contract also receives withdrawal messages from L2 with the number of wstETH tokens to unlock. The contract is upgradeable and permissioned, withdrawals and deposits can be paused.

L2ERC20ExtendedTokensBridge

The contract is a counterpart of the `L1LidoTokensBridge` contract on L2. It receives messages from L1, mints wstETH tokens, wraps them into stETH tokens, and sends them to the user. In the case of withdrawals from L2 to L1, it can unwrap stETH tokens into wstETH, burn wstETH and send a message to L1. As the `L1LidoTokensBridge` contract, it is upgradeable and permissioned. Withdrawals and deposits can be paused.

ERC20BridgedPermit

The contract represents the wstETH token on L2. It is an ERC20 token with additional permit functionality. It can be minted and burned by the `L2ERC20ExtendedTokensBridge` contract.

ERC20RebasableBridgedPermit

The contract represents the stETH token on L2. It is a rebasable ERC20 token with additional permit functionality. The internal balance is represented by shares, and the actual number of tokens is computed based on the current rate from the `TokenRateOracle` contract. New stETH tokens can be created by wrapping the wstETH tokens.

Actors

This part describes actors of the system, their roles, and permissions.

Bridge Admin

The address that is granted the `DEFAULT_ADMIN_ROLE` role in the `L1LidoTokensBridge` and `L2ERC20ExtendedTokensBridge` contracts. The role allows the admin to grant and revoke roles.

Proxy Admin

The address that can upgrade the implementation to a new version and ossify the proxy.

Deposit Disabler

The address that can pause and unpause deposits in the `L1LidoTokensBridge` and `L2ERC20ExtendedTokensBridge` contracts.

Withdrawal Disabler

The address that can pause and unpause withdrawals in the `L1LidoTokensBridge` and `L2ERC20ExtendedTokensBridge` contracts.

Optimism Messenger

The relayer that sends messages between L1 and L2. It is a part of the Optimism stack.

User

The address that interacts with the system by depositing and withdrawing tokens.

5.2. Trust Model

Don't trust, verify.

The system is designed to be permissioned. The bridge contracts are upgradeable, and the users need to trust the proxy admin not to upgrade the contracts to a malicious version. Also, the bridges have roles that allow pausing deposits and withdrawals. If these functionalities are disabled during the initiated deposit and withdrawal before the message is delivered to the destination chain, the tokens will be locked in the contract on the source chain until the bridges are enabled again and the message is manually replayed. Also, the users need to rely on the up-to-date token rate on L2 to have confidence in having the correct number of stETH tokens.

Last but not least, the system relies on the Optimism stack, including the sequencer and the implementation of fault proofs. If messages are not delivered to L2, the token rate may become outdated, which may create an arbitrage opportunity. If the sequencer behaves maliciously, it may censor messages or create invalid blocks. The fault proofs mechanism on Optimism is not yet implemented on the mainnet as of the audit date.

L1: Insufficient token rate precision

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	`**/*`	Type:	Arithmetics

Description

The project is responsible for bridging stETH and wstETH tokens and the tokens/shares conversion rate.

The token rate is computed as `10^18 * stETH.getTotalPooledEther() / stETH.getTotalShares()` on the Ethereum mainnet and bridged to Optimism.

To compute the amount of tokens based on the amount of shares on Optimism, the following code is used:

Listing 1. Excerpt from [ERC20RebasableBridged]()

```
    function _getTokensByShares(uint256 sharesAmount_) internal view returns
(uint256) {
        (uint256 tokensRate, uint256 decimals) = _getTokenRateAndDecimal();
        return (sharesAmount_ * tokensRate) / (10 ** decimals);
    }
```

Due to the division performed while computing the token rate on the mainnet and the limited precision of the rate, the value returned from `_getTokensByShares` may be off by a small amount.

Exploit scenario

A user bridges 1000 * 10^18 stETH tokens from the Ethereum mainnet to Optimism. The correct amount of shares is bridged, but due to the limited precision of the token rate, the reported `balanceOf` is lower by 52 wei than the expected value.

Recommendation

Increase the token rate precision by using a higher precision factor, e.g., `10^27`, or pass both the total pooled ether and total shares to Optimism.

Fix 1.1

The codebase was refactored. There are several major changes in the codebase. First, the type member `rate` of the `DepositData` structure in the `DepositDataCodec` contract was changed to `uin128` from `uint96`. The newly added `TokenRateAndUpdateTimestampProvider` now provides the token rate, the latest update timestamp and the token rate decimals, which are hardcoded to the value `27`. On L2, The `TokenRateOracle` contract was largely refactored, including work with newly added 27 decimals for the token rate.

[Go back to Findings Summary]

L2: `unwrap` inconsistent tokens amount in event

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	ERC20RebasableBridged.sol	Type:	Arithmetics

Description

The functions `unwrap` and `bridgeUnwrap` convert stETH tokens to wstETH tokens. Both functions accept the stETH tokens amount that is converted to the amount of shares. In order to emit the ERC-20 `Transfer` event, the shares amount is converted back to the stETH tokens amount.

However, due to roundings and divide-before-multiply data dependency, the input amount of tokens and the amount reported in the `Transfer` event may be different, posing an inconsistency.

See Appendix C for the full data dependency trace.

Exploit scenario

A user calls `unwrap` with `764035550674393190` as the input amount of tokens. The `Transfer` event contains `764035550674393188` as the value and the difference in `balanceOf` before the transaction and after the transaction is `764035550674393189`.

Recommendation

Consider emitting the `Transfer` event with the same amount of tokens as the input amount or as the real `balanceOf` change.

Fix 1.1

The code of `ERC20RebasableBridged` was refactored to prevent the issue. The _unwrap function now calculates the number of shares from the token amount provided in the input and calls the newly introduced _unwrapShares function with the token and share amounts:

Listing 2. Excerpt from [ERC20RebasableBridged]()

```
    function _unwrap(address account_, uint256 tokenAmount_) internal returns (uint256) {
        if (tokenAmount_ == 0) revert ErrorZeroTokensUnwrap();
        uint256 sharesAmount = _getSharesByTokens(tokenAmount_);
        return _unwrapShares(account_, sharesAmount, tokenAmount_);
    }
```

The _unwrapShares function then burns the computed number of shares and emits the transfer events with the token and share amounts obtained from the _unwrap function:

Listing 3. Excerpt from [ERC20RebasableBridged]()

```
    function _unwrapShares(address account_, uint256 sharesAmount_, uint256 tokenAmount_) internal returns (uint256) {
        if (sharesAmount_ == 0) revert ErrorZeroSharesUnwrap();
        _burnShares(account_, sharesAmount_);
        _emitTransferEvents(account_, address(0), tokenAmount_, sharesAmount_);
        TOKEN_TO_WRAP_FROM.safeTransfer(account_, sharesAmount_);
        return sharesAmount_;
    }
```

This flow ensures that the token and share amounts are consistent in the events.

[Go back to Findings Summary]()

W1: Usage of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler configuration

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Exploit scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Fix 1.1

The issue was acknowledged with the comment:

> There are already proxies that were compiled and deployed with Solc.

Go back to Findings Summary

W2: ERC-20 `transferFrom` emits `Approval`

Impact:	Warning	Likelihood:	N/A
Target:	ERC20Core.sol, ERC20RebasableBridged.sol	Type:	Non-standard tokens

Description

Both implementations `ERC20Core` and `ERC20RebasableBridged` emit the `Approval` event when calling `transferFrom`. This is uncommon and may confuse off-chain logic.

Recommendation

Consider not emitting the `Approval` event when calling `transferFrom`.

Fix 1.1

The issue was acknowledged with the comment:

> Core protocol also emits those events.

Go back to Findings Summary

W3: False comments

Impact:	Warning	Likelihood:	N/A
Target:	IL2ERC20Bridge.sol	Type:	Code quality

Description

The codebase contains the following false comments:

Listing 4. Excerpt from [IL2ERC20Bridge]()

```
    /// @param l1Gas_ Unused, but included for potential forward
compatibility considerations.
```

Listing 5. Excerpt from [IL2ERC20Bridge]()

```
    /// @param l1Gas_ Unused, but included for potential forward
compatibility considerations.
```

The comments state that the `l1Gas_` parameter is unused. However, it is used in the code.

Recommendation

Fix the false comments.

Fix 1.1

The comments were changed to reflect the valid use of the `l1Gas_` parameter.

[Go back to Findings Summary]()

W4: Limited ERC-2612 use-case with ERC-1271

Impact:	Warning	Likelihood:	N/A
Target:	PermitExtension.sol	Type:	ERC incompatibility

Description

The ERC-2612 `permit` signature is:

```
function permit(address owner, address spender, uint value, uint deadline, uint8
v, bytes32 r, bytes32 s) external;
```

When `owner` is a contract, the `r`, `s`, `v` components are concatenated together and sent to the `owner` contract as the `_signature` parameter:

```
function isValidSignature(bytes32 _hash, bytes memory _signature) public view
returns (bytes4 magicValue);
```

However, some ERC-1271 contracts may require more than 65 bytes to verify a signature.

Recommendation

There is an ongoing discussion about extending the permit ERC-2612 with additional function:

```
function permit(address owner, address spender, uint value, uint deadline, bytes
memory signature) external;
```

Consider implementing this extension to allow arbitrary-length signatures.

Fix 1.1

In addition to the existing signature, a new `permit()` function was added to the `PermitExtension` contract:

Listing 6. Excerpt from [PermitExtension]

```
        address owner_,
        address spender_,
        uint256 value_,
        uint256 deadline_,
        bytes calldata signature_
    ) external {
        _permit(owner_, spender_, value_, deadline_, signature_);
    }
```

This additional function fixes the issue. However, the code now uses the `isValidSignatureNow()` function from OpenZeppelin contracts v4.6.0, which is [known] to be vulnerable to reverting for specific signers. The code is not affected by this vulnerability, since the logic is expected to revert if the signature is invalid, however, we still warn about this issue for potential future changes.

[Go back to Findings Summary]

W5: Use of a deprecated function

Impact:	Warning	Likelihood:	N/A
Target:	BridgingManager.sol	Type:	Deprecated function

Description

In the `_initializeBridgingManager()` function of the `BridgingManager` contract, the `_setupRole()` function from OpenZeppelin `AccessControl` contract is used to grant the admin role:

Listing 7. Excerpt from BridgingManager

```
        _setupRole(DEFAULT_ADMIN_ROLE, admin_);
```

However, the documentation for `_setupRole()` states that it "is deprecated in favor of `_grantRole`."

Recommendation

Use the `_grantRole()` function instead:

```
_grantRole(DEFAULT_ADMIN_ROLE, admin_);
```

Fix 1.1

The `_grantRole()` function is now used instead of `_setupRole()`.

Go back to Findings Summary

ackee | blockchain security

W6: Initializers can be front-run

Impact:	Warning	Likelihood:	N/A
Target:	TokenRateOracle.sol, L1LidoTokensBridge.sol, L2ERC20ExtendedTokensBridge.sol, ERC20BridgedPermit.sol, ERC20RebasableBridgedPermit.sol	Type:	Upgradeability

Description

Bridges and tokens contracts are expected to be hidden behind proxies. Thus, the contract initialization process has three steps: contract deployment, a call to the `initialize` function through the proxy, and the change of the implementation address in the proxy contract. Without using a factory contract, these operations are not atomic, and there is a risk of the initialization front-running if the initializers are not properly protected. This is the case for the following contracts and their initializers:

- `TokenRateOracle`: The malicious initialization of the oracle may lead to a wrong initial token rate set.
- `L1LidoTokensBridge`: The front-run initialization may lead to granting the admin role to a malicious address.
- `L2ERC20ExtendedTokensBridge`: The front-run initialization may lead to granting the admin role to a malicious address.
- `ERC20BridgedPermit`: The front-run initialization may lead to setting incorrect token metadata.
- `ERC20RebasableBridgedPermit`: The front-run initialization may lead to

setting incorrect token metadata.

Recommendation

To prevent the front-running of the initialization transaction, consider the following measures:

- Protect the initializer functions with a modifier that restricts access to them;
- Ensure the upgrades are performed using the `proxy__upgradeToAndCall()` function on the proxy contract to ensure atomicity;
- Make sure the deployment script detects failed initializations and redeploys the contracts.

Fix 1.1

The issue was acknowledged with the comment:

> The deployment with `upgradeAndCall` solves the problem.

Go back to Findings Summary

W7: Linear calculation of the allowed token rate deviation

Impact:	Warning	Likelihood:	N/A
Target:	TokenRateOracle.sol	Type:	Arithmetics

Description

In the TokenRateOracle contract, the new rate is accepted if it is in the allowed range. The range is computed as the deviation from the current rate, which is the allowed rate percentage change per day times the number of days passed since the last rate update, rounded up:

Listing 8. Excerpt from TokenRateOracle

```
        uint256 rateL1TimestampDiff = newRateL1Timestamp_ - _getRateL1Timestamp();
        uint256 roundedUpNumberOfDays = rateL1TimestampDiff / ONE_DAY_SECONDS + 1;
        uint256 allowedTokenRateDeviation = roundedUpNumberOfDays * MAX_ALLOWED_TOKEN_RATE_DEVIATION_PER_DAY;
```

This approach, however, is an approximation of the correct compound percentage formula. If the rate changes, for example, by one percent per day for three days, the overall change will not be 3% but rather `1.01^3`%, or ~3.03%. The error highly depends on the magnitude of `MAX_ALLOWED_TOKEN_RATE_DEVIATION_PER_DAY` and the number of days passed since the last rate update. For larger values of `MAX_ALLOWED_TOKEN_RATE_DEVIATION_PER_DAY` and more days passed, the error will be more significant.

Recommendation

Consider using the correct compound percentage formula to calculate the allowed rate deviation or ensure that the error introduced by the current approach is acceptable.

Fix 1.1

The issue was acknowledged.

[Go back to Findings Summary]

ackee | blockchain security

W8: Insufficient data validation

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Data validation

Description

Multiple contracts have insufficient data validation for parameters that are passing addresses in their constructors or initializers. The following contracts are affected by a lack of checks against the zero value:

- `CrossDomainEnabled`: `messenger_` in the constructor.
- `L1ERC20ExtendedTokensBridge`: `l2TokenBridge_` in the constructor.
- `L1LidoTokensBridge`: `admin_` in the initializer.
- `L2ERC20ExtendedTokensBridge`: `l1TokenBridge_` in the constructor and `admin_` in the initializer.
- `OpStackTokenRatePusher`: `wstEth_`, `tokenRateOracle_`, and `l2GasLimitForPushingTokenRate_` in the constructor.
- `RebasableAndNonRebasableTokens`: `l1TokenNonRebasable_`, `l1TokenRebasable_`, `l2TokenNonRebasable_`, and `l2TokenRebasable_` in the constructor.
- `TokenRateOracle`: `l2ERC20TokenBridge_`, `l1TokenRatePusher_`, `tokenRateOutdatedDelay_`, `maxAllowedL2ToL1ClockLag_`, `maxAllowedTokenRateDeviationPerDay_` in the constructor and `tokenRate_` and `rateL1Timestamp_` in the initializer.
- `ERC20Bridged`: `bridge_` and `decimals_` in the constructor.
- `ERC20Metadata`: `decimals_` in the constructor.
- `ERC20RebasableBridged`: `tokenToWrapFrom_`, `tokenRateOracle_` and `l2ERC20TokenBridge_` in the constructor.

Recommendation

Add zero-value check for all mentioned parameters.

Fix 1.1

Missing checks listed above were added. The issue is fixed.

[Go back to Findings Summary]

ackee blockchain security

I1: Uncached `.length` in for loop

Impact:	Info	Likelihood:	N/A
Target:	TokenRateNotifier.sol	Type:	Gas optimization

Description

In the following code snippets, `.length` of an array is used in a for loop without modifying the array:

Listing 9. Excerpt from TokenRateNotifier

```
        for (uint256 obIndex = 0; obIndex < observers.length; obIndex++) {
```

Listing 10. Excerpt from TokenRateNotifier

```
        for (uint256 obIndex = 0; obIndex < observers.length; obIndex++) {
```

Recommendation

Cache the length of the array to save gas.

Fix 1.1

The length is now cached before entering the loop.

Go back to Findings Summary

I2: Inconsistent modifiers order

Impact:	Info	Likelihood:	N/A
Target:	L1ERC20ExtendedTokensBridge.sol, L2ERC20ExtendedTokensBridge.sol	Type:	Code quality

Description

The `finalizeERC20Withdrawal` and `finalizeDeposit` functions are called on the destination chain of a cross-chain transfer to finalize the transfer.

Listing 11. Excerpt from [L1ERC20ExtendedTokensBridge]()

```
    function finalizeERC20Withdrawal(
        address l1Token_,
        address l2Token_,
        address from_,
        address to_,
        uint256 amount_,
        bytes calldata data_
    )
        external
        whenWithdrawalsEnabled
        onlyFromCrossDomainAccount(L2_TOKEN_BRIDGE)
        onlySupportedL1L2TokensPair(l1Token_, l2Token_)
```

Listing 12. Excerpt from [L2ERC20ExtendedTokensBridge]()

```
    function finalizeDeposit(
        address l1Token_,
        address l2Token_,
        address from_,
        address to_,
        uint256 amount_,
        bytes calldata data_
    )
        external
```

```
        whenDepositsEnabled()
        onlySupportedL1L2TokensPair(l1Token_, l2Token_)
        onlyFromCrossDomainAccount(L1_TOKEN_BRIDGE)
```

Both functions apply the analogous modifiers for the same purpose, but the order of the modifiers is different.

Recommendation

Consider unifying the modifiers order to achieve the same behavior on both chains.

Fix 1.1

The issue was fixed by changing the order of the modifiers in the `L2ERC20ExtendedTokensBridge` contract.

Go back to Findings Summary

I3: Unused code

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Code quality

Description

The project contains multiple occurrences of unused code. See Appendix C for the full list.

Recommendation

Remove the unused code to improve the readability and maintainability of the codebase.

Fix 1.1

The unused errors and events were removed from the codebase.

Go back to Findings Summary

I4: Typos

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Code quality

Description

There are multiple typos in the codebase.

Space before comma:

Listing 13. Excerpt from [ERC20RebasableBridged]()

```
        _emitTransferEvents(from_, to_, amount_ ,sharesToTransfer);
```

Bad indentation:

Listing 14. Excerpt from [ERC20RebasableBridged]()

```
    function _wrap(address from_, address to_, uint256 sharesAmount_)
internal returns (uint256) {
```

Typo:

Listing 15. Excerpt from [ERC20RebasableBridged]()

```
    error ErrorTrasferToRebasableContract();
```

Recommendation

Fix the typos.

Fix 1.1

All problems listed above were fixed.

Go back to Findings Summary

ackee blockchain security

I5: _mintShares can return tokensAmount to save gas

Impact:	Info	Likelihood:	N/A
Target:	ERC20RebasableBridged	Type:	Gas optimization

Description

In the ERC20RebasableBridged contract, the _wrap function calls the _mintShares function, which calls _getTokensByShares to emit the event with the minted token amount. However, the _wrap function also returns the token amount calculated by the second call to _getTokensByShares. _getTokensByShares performs arithmetic calculations and performs two external calls to the oracle.

Listing 16. Excerpt from [ERC20RebasableBridged._mintShares]()

```
    function _mintShares(
        address recipient_,
        uint256 amount_
    ) internal onlyNonZeroAccount(recipient_) {
        _setTotalShares(_getTotalShares() + amount_);
        _getShares()[recipient_] = _getShares()[recipient_] + amount_;
        uint256 tokensAmount = _getTokensByShares(amount_);
        _emitTransferEvents(address(0), recipient_, tokensAmount ,amount_);
    }
```

Listing 17. Excerpt from [ERC20RebasableBridged._wrap]()

```
    function _wrap(address from_, address to_, uint256 sharesAmount_)
internal returns (uint256) {
        if (sharesAmount_ == 0) revert ErrorZeroSharesWrap();

        TOKEN_TO_WRAP_FROM.safeTransferFrom(from_, address(this),
sharesAmount_);
        _mintShares(to_, sharesAmount_);

        return _getTokensByShares(sharesAmount_);
```

```
372     }
```

The second call can be avoided if the `_mintShares` function returns the token amount to the `_wrap` function. This approach can save gas.

Recommendation

Consider changing the signature of the `_mintShares` function to return the token amount.

Fix 1.1

The issue was acknowledged with the comment:

> The function name won't fit the return value.

[Go back to Findings Summary]

6. Report revision 1.1

6.1. System Overview

The codebase was modified to include fixes for the reported issues, several gas optimizations and a complete refactoring of the `TokenRateOracle` contract. Additionally, a new contract was added for use on L1.

Contracts

The `TokenRateOracle` contract was refactored and now includes two new roles allowing to pause and resume token rate updates. It also has limits on the maximum and minimum token rate and stores historical token rate updates in an array (previously it stored only the last update).

The newly introduced `TokenRateAndUpdateTimestampProvider` contract is used on L1 to provide the current token rate and the timestamp of the last update. The timestamp is now calculated from the genesis block timestamp and the block number of the last update. The token rate decimals were also increased from 18 to 27.

Last but not least, the `ERC20RebasableBridged` contract was also refactored to include several optimizations and the new `unwrapShares` function in addition to the existing `unwrap` function.

Actors

The `TokenRateOracle` contract now includes two new roles: `RATE_UPDATE_ENABLER_ROLE` and `RATE_UPDATE_DISABLER_ROLE` for pausing and resuming token rate updates.

6.2. Trust Model

If the token rate updates are paused, the `TokenRateOracle` contract will emit the event and return. This may prevent the correct update of the token rate on L2 and create a discrepancy between the token rate on L1 and L2, causing arbitrage opportunities. Users must trust the entities with the newly added roles to not pause the token rate updates for an extended period.

7. Report revision 1.2

7.1. System Overview

The codebase was modified to include fixes for the issues W8 and I4 that were partially fixed in the previous revision. No other changes were made to the codebase.

Actors

No changes were made to the actors in the system.

7.2. Trust Model

No changes were made to the trust model.

8. Report revision 1.3

8.1. System Overview

The audited scope includes all the changes in the protocol since the last revision.

Contracts

This section contains an outline of the audited contracts that we find important for the review.

TokenRateAndUpdateTimestampProvider

An abstract contract that provides the token rate and the L1 timestamp of the last update. The contract is expected to be deployed on L1, and the token rate is taken directly from the wstETH contract. The timestamp is calculated as the sum of the genesis block timestamp and the product of the number of slots since the genesis block and the slot duration. The slot number is taken from the accounting oracle contract, which is passed as a parameter to the contract constructor and is outside the scope of the protocol. Second, the contract now throttles the rate updates to once per a certain number of seconds.

TokenRateOracle

The contract was largely refactored since the last revision. The oracle is deployed on L2 and is responsible for providing the token rates and timestamps of all token rate updates. In comparison to the previous revision, the contract now stores all historical updates, not only the last one. Also, several new requirements for the input data were added to the code. First, the token rate should now be between minimum and maximum sane values.

In addition to the changes mentioned above, the contract can now be paused and resumed. During the pause, a historical rate can be set with the limitation that the rate should not be older than a certain number of seconds. This allows the contract to invalidate an incorrectly set token rate in cases of emergency and pause all rate updates until the issue is resolved. Pausing can be done by the Lido committee with the `RATE_UPDATE_DISABLER_ROLE` role while resuming is done by the Lido DAO with the `RATE_UPDATE_ENABLER_ROLE` role.

TokenRateNotifier

The contract remained unchanged since the last revision, however, the `handlePostTokenRebase` function can now be called only by the Lido contract that is passed as a parameter to the contract constructor.

Actors

This part describes changes to the actors of the system, their roles, and permissions.

Accounting Oracle

The oracle provides the slot number to the `TokenRateAndUpdateTimestampProvider` contract. The correctness of the slot number is crucial for the correct calculation of the timestamp.

RATE_UPDATE_DISABLER_ROLE

The role is assigned to the Lido committee and allows pausing token rate updates in the `TokenRateOracle` contract.

RATE_UPDATE_ENABLER_ROLE

The role is assigned to the Lido DAO and allows resuming token rate updates in the `TokenRateOracle` contract.

Lido Contract

The Lido contract is now allowed to call the `handlePostTokenRebase` function in the `TokenRateNotifier` contract.

8.2. Trust Model

The trust model remains almost the same as in the previous revision. Users must trust the Lido DAO and the Lido committee to keep the token rates up-to-date and correct. The accounting oracle must provide the correct slot number to the `TokenRateAndUpdateTimestampProvider` contract.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Lido: stETH on Optimism, 18.06.2024.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

ackee | blockchain security

Appendix C: Wake outputs

This section lists the outputs from the Wake tool used during the audit.

C.1. Detectors

wake detect unused-event

```
[INFO][HIGH] Unused event [unused-event]
        bytes _data
    );

  ❱     event DepositFailed(
        address indexed _l1Token,
        address indexed _l2Token,

contracts/optimism/interfaces/IL2ERC20Bri
```

Figure 1. Unused events

wake detect unused-error

```
[INFO][HIGH] Unused error [unused-error]
            _;
        }

  ❱     error ErrorUnsupportedL1Token(address l1Token);
        error ErrorUnsupportedL2Token(address l2Token);
        error ErrorUnsupportedL1L2TokensPair(address l1Token, address l2Token);

contracts/optimism/RebasableAndNonRebasableTokens.sol

[INFO][HIGH] Unused error [unused-error]
        error ErrorNotEnoughBalance();
        error ErrorNotEnoughAllowance();
        error ErrorAccountIsZeroAddress();
  ❱     error ErrorDecreasedAllowanceBelowZero();
    }

contracts/token/ERC20Core.sol

[INFO][HIGH] Unused error [unused-error]
        error ErrorNotEnoughBalance();
        error ErrorNotEnoughAllowance();
        error ErrorAccountIsZeroAddress();
  ❱     error ErrorDecreasedAllowanceBelowZero();
        error ErrorNotBridge();
    }

contracts/token/ERC20RebasableBridged.sol
```

Figure 2. Unused errors

wake detect divide-before-multiply

```
┌ [MEDIUM][MEDIUM] Divide before multiply [divide-before-multiply] ─────────────────
│
│    function _getSharesByTokens(uint256 tokenAmount_) internal view returns (uint256) {
│        (uint256 tokensRate, uint256 decimals) = _getTokenRateAndDecimal();
│ ❱      return (tokenAmount_ * (10 ** decimals)) / tokensRate;
│    }
│
│
└ contracts/token/ERC20RebasableBridged.sol ─────────────────────────────────────────
└── ┌ Data dependency ──────────────────────────────────────────────────────────────────
    │    function _unwrap(address account_, uint256 tokenAmount_) internal returns (uint2
    │        if (tokenAmount_ == 0) revert ErrorZeroTokensUnwrap();
    │
    │ ❱      uint256 sharesAmount = _getSharesByTokens(tokenAmount_);
    │        _burnShares(account_, sharesAmount);
    │        TOKEN_TO_WRAP_FROM.safeTransfer(account_, sharesAmount);
    │
    └ contracts/token/ERC20RebasableBridged.sol ─────────────────────────────────────
    └── ┌ Data dependency ──────────────────────────────────────────────
        │        if (tokenAmount_ == 0) revert ErrorZeroTokensUnwrap();
        │
        │        uint256 sharesAmount = _getSharesByTokens(tokenAmount_);
        │ ❱      _burnShares(account_, sharesAmount);
        │        TOKEN_TO_WRAP_FROM.safeTransfer(account_, sharesAmount);
        │
        │
        └ contracts/token/ERC20RebasableBridged.sol ──────────────────────
        └── ┌ Data dependency ───────────────────────────────────────────────────────
            │        if (accountShares < amount_) revert ErrorNotEnoughBalance();
            │        _setTotalShares(_getTotalShares() - amount_);
            │        _getShares()[account_] = accountShares - amount_;
            │ ❱      uint256 tokensAmount = _getTokensByShares(amount_);
            │        _emitTransferEvents(account_, address(0), tokensAmount ,amount_);
            │    }
            │
            └ contracts/token/ERC20RebasableBridged.sol ──────────────────────────
            └── ┌ Multiply ────────────────────────────────────────────────────────
                │
                │    function _getTokensByShares(uint256 sharesAmount_) internal view ret
                │        (uint256 tokensRate, uint256 decimals) = _getTokenRateAndDecimal
                │ ❱      return (sharesAmount_ * tokensRate) / (10 ** decimals);
                │    }
                │
                │
                └ contracts/token/ERC20RebasableBridged.sol ─────────────────────
```

Figure 3. Divide-before-multiply occurence

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://twitter.com/AckeeBlockchain