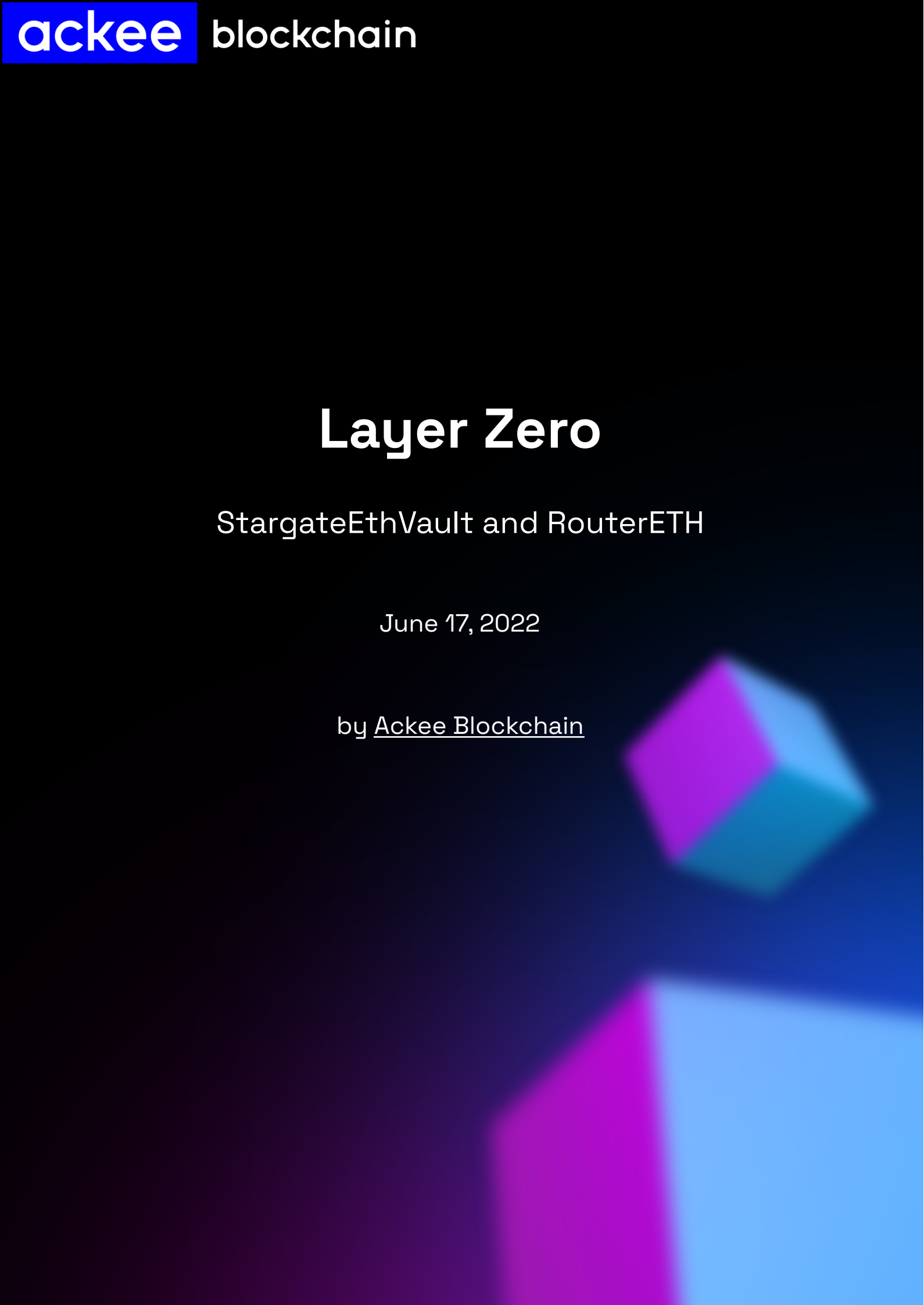
ackee blockchain

Layer Zero

StargateEthVault and RouterETH

June 17, 2022

by Ackee Blockchain



Contents

1. Document Revisions

1.0	Final report	Jun 17, 2022
1.1	Fix review	Jun 22, 2022

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, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course 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, Rockaway Blockchain Fund.

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 Slither 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 fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Review team

Member's Name	Position
Jan Šmolík	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. 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

Layer Zero engaged Ackee Blockchain to conduct a security review of the `StargateEthVault` and `RouterETH` contracts with a total time donation of three engineering days. The review took place between June 14 and June 17, 2022.

The commit we worked on is the following:

- `8d0b07ad326c77d749b0f67001af7d86c56d9a64`

`StargateEthVault` is a fork of `WETH9.sol` with minor changes and `RouterETH` is just a wrapper around two functions of another Layer Zero's contract, so there is not much space for a critical problem. The review resulted in nine findings, ranging from Informational to Medium severity.

Ackee Blockchain recommends Layer Zero to address all of the issues discussed in this report and give us feedback, so that the report could be updated.

Update June 22, 2022: Layer Zero provided an updated codebase that addresses the issues from this report. See Appendix B for a detailed discussion of the exact status of each issue.

Although the project still uses the `solc` optimizer (M1), all other issues identified in this report were fixed.

4. 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.

4.1. Contracts

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

StargateEthVault.sol

StargateEthVault is a fork of the `WETH9.sol` contract (i.e., ERC-20 compliant Wrapped Ether).

The difference is that it automatically unwraps to the native gas token on transfers. The owner of the contract is able to disable this auto-unwrap on transfers to certain addresses.

RouterETH.sol

RouterETH is a wrapper around the `addLiquidity()` and `swap()` functions of the Layer Zero's `Router.sol`.

4.2. Actors

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

StargateEthVault owner

The owner of the StargateEthVault contract. He can:

- transfer or renounce the ownership of the contract;

- call the `setNoUnwrapTo()` function to disable auto-unwrap on transfers to certain addresses.

4.3. Trust model

There are no trust issues in these two contracts.

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact 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 *Informational*.

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

5.1. Finding classification

The full definitions are as follows:

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 "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Informational

The issue is on the border-line 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.

6. Findings

This section contains the list of discovered findings. Unless overriden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Type	Impact	Likelihood
M1: Usage of `solc` optimizer	Compiler configuration	High	Low
L1: Insufficient data validation in RouterETH	Data validation	High	Low
W1: `totalSupply()` not guaranteed to be accurate	Code logic	Warning	N/A
W2: Renounce ownership	Access controls	Warning	N/A
I1: Public functions	Gas optimization	Info	N/A
I2: Variables should be declared `constant`	Gas optimization	Info	N/A
I3: Unused imports	Code quality	Info	N/A
I4: Comments quality	Code quality	Info	N/A

	Type	Impact	Likelihood
I5: Code quality	Code quality	Info	N/A

Table 1. Table of Findings

M1: Usage of `solc` optimizer

Impact:	High	Likelihood:	Low
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.

Vulnerability 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.

Go back to Findings Summary

L1: Insufficient data validation in RouterETH

Impact:	High	Likelihood:	Low
Target:	RouterETH	Type:	Data validation

Description

RouterETH does not perform any data validation of the following passed addresses in its constructor:

- `_stargateEthVault`
- `_stargateRouter`

Exploit scenario

An incorrect or malicious `_stargateEthVault` is passed to the constructor. Instead of reverting, the call succeeds.

Recommendation

Add more stringent data validation for `_stargateEthVault` and `_stargateRouter`. At least, this should include a zero-address check.

Go back to Findings Summary

W1: `totalSupply()` not guaranteed to be accurate

Impact:	Warning	Likelihood:	N/A
Target:	StargateEthVault	Type:	Code logic

Description

In StargateEthVault, the total supply of `SGETH` is measured by the amount of `ETH` in the contract.

```
function totalSupply() public view returns (uint) {
        return address(this).balance;
    }
```

This makes sense, because each time someone deposits or sends `ETH` to the contract, `SGETH` is added to his balance, and each time someone withdraws `ETH`, his `SGETH` balance is reduced accordingly.

However, there are two cases where ether can exist in the contract without having executed the `deposit()` function. Therefore, `totalSupply()` could return a larger number than the actual sum of `SGETH` balances.

Self-destruct ether

Any contract is able to implement the `selfdestruct()` function, which removes all bytecode from the contract address and sends all ether stored there to the parameter-specified address. If this specified address is also a contract, no functions (including the `fallback()`) get called. Therefore, the `selfdestruct()` function can be used to forcibly send ether to this contract.

Pre-sent ether

Another way to get ether into the contract is to preload the contract address with ether, because the contract address is deterministic.

Recommendation

If having an accurate `totalSupply()` is important, add some logic to track the supply accurately (i.e., increasing the supply in `deposit()` and lowering it in `withdraw()` and sometimes in `transferFrom()`). Otherwise, just keep this in mind.

Go back to Findings Summary

W2: Renounce ownership

Impact:	Warning	Likelihood:	N/A
Target:	StargateEthVault	Type:	Access controls

Description

The StargateEthVault is an `Ownable` contract. The owner of the contract can call the `setNoUnwrapTo()` function. The ownership can be transferred and renounced by the owner.

Exploit scenario

The owner accidentally calls `renounceOwnership()`.

Recommendation

We recommend overriding the `renounceOwnership()` method to disable this feature if it is not intended. Otherwise, ignore this issue.

Go back to Findings Summary

I1: Public functions

Impact:	Informational	Likelihood:	N/A
Target:	StargateEthVault	Type:	Gas optimization

Description

The following functions are declared public even though they are not called internally anywhere:

- `withdraw()`
- `totalSupply()`
- `approve()`
- `transfer()`

Recommendation

If functions are not called internally, they should be declared external. It helps gas optimization because function arguments do not have to be copied into memory.

Go back to Findings Summary

I2: Variables should be declared `constant`

Impact:	Informational	Likelihood:	N/A
Target:	StargateEthVault	Type:	Gas optimization

Description

In StargateEthVault, the values of the following variables are fixed at compile-time and can never change:

- `name`
- `symbol`
- `decimals`

Recommendation

Declare these variables `constant`. Compared to regular state variables, the gas costs of `constant` variables are much lower.

Go back to Findings Summary

I3: Unused imports

Impact:	Informational	Likelihood:	N/A
Target:	RouterETH	Type:	Code quality

Description

In RouterETH, the following imports are unnecessary:

```
import "./openzeppelin/Ownable.sol";
import "./openzeppelin/IERC20.sol";
```

Recommendation

Remove these imports.

Go back to Findings Summary

I4: Comments quality

Impact:	Informational	Likelihood:	N/A
Target:	*	Type:	Code quality

Description

There are a few minor things in the comments that should be fixed.

setNoUnwrapTo comment

In StargateEthVault, the comment above the `setNoUnwrapTo()` function reflects the old version of the function when it had two parameters. Now, you only call `setNoUnwrapTo( addr )`.

```
// if you do NOT wish to unwrap eth on transfers TO
// certain addresses, call `setNoUnwrapTo( addr, true )`
```

SGWETH

Also, in `transferFrom()`, there is a small mistake in the `require` error message. The symbol is not `SGWETH`, but `SGETH`.

```
require(success, "SGWETH: failed to transfer");
```

WETH

In RouterETH, `WETH` is often used for referencing `SGETH` in the comments.

Recommendation

Consider fixing this so that the codebase is cleaner.

Go back to Findings Summary

I5: Code quality

Impact:	Informational	Likelihood:	N/A
Target:	RouterETH	Type:	Code quality

Description

1. The `IStargateEthVault` interface is declared in the RouterETH.sol file. StargateEthVault does not implement this interface.
2. In RouterETH, there are two variables for communication with other contracts:

- `address public immutable stargateEthVault`
- `IStargateRouter public immutable stargateRouter`

Recommendation

1. Declare `IStargateEthVault` in an independent file and import it both in `StargateEthVault` and `RouterETH`. `StargateEthVault` should implement it (keyword `is`). This reduces the space for human errors in future adjustments.
2. Consider making the variables consistent for a cleaner codebase, that is:

- `IStargateEthVault public immutable stargateEthVault`
- `IStargateRouter public immutable stargateRouter`

or

- `address public immutable stargateEthVault`
- `address public immutable stargateRouter`

Go back to Findings Summary

7. Appendix A

7.1. How to cite

Please cite this document as:

Ackee Blockchain, Layer Zero Stargate Router Eth, June 17, 2022.

If an individual issue is referenced, please use the following identifier:

`ABCH-{project_identifer}-{finding_id}`,

where `{project_identifier}` for this project is `LAYER-ZERO-STARGATE-ROUTER-ETH` and `{finding_id}` is the id which can be found in Summary of Findings. For example, to cite I2 issue, we would use `ABCH-LAYER-ZERO-STARGATE-ROUTER-ETH-I2`.

8. Appendix B: Fix Review

On June 22, 2022, Ackee Blockchain reviewed Layer Zero's fixes for the issues identified in this report. The following table summarizes the fix review.

Fix log

Id		Type	Impact	Likelihood	Status
M1F	M1F: Usage of `solc` optimizer	Compiler configuration	High	Low	**Acknowledged**
L1F	L1F: Insufficient data validation in RouterETH	Data validation	High	Low	**Fixed**
W1F	W1F: `totalSupply()` not guaranteed to be accurate	Code logic	Warning	N/A	**Fixed**
W2F	W2F: Renounce ownership	Access controls	Warning	N/A	**Fixed**
I1F	I1F: Public functions	Gas optimization	Info	N/A	**Fixed**
I2F	I2F: Variables should be declared `constant`	Gas optimization	Info	N/A	**Fixed**
I3F	I3F: Unused imports	Code quality	Info	N/A	**Fixed**
I4F	I4F: Comments quality	Code quality	Info	N/A	**Fixed**

Id		Type	Impact	Likelihood	Status
I5F	I5F: Code quality	Code quality	Info	N/A	**Fixed**

Table 2. Table of fixes

M1F: Usage of `solc` optimizer

Impact:	High	Likelihood:	Low
Target:	M1: Usage of `solc` optimizer	Type:	Compiler configuration

Description

The project still uses the `solc` optimizer.

Go back to Fix log

L1F: Insufficient data validation in RouterETH

Impact:	High	Likelihood:	Low
Target:	L1: Insufficient data validation in RouterETH	Type:	Data validation

Description

RouterETH now performs zero-address checks in its constructor of the following arguments, as was recommended:

- `_stargateEthVault`
- `_stargateRouter`

```
constructor(address _stargateEthVault, address _stargateRouter, uint16
_poolId){
        require(_stargateEthVault != address(0x0), "RouterETH:
_stargateEthVault cant be 0x0");
        require(_stargateRouter != address(0x0), "RouterETH:
_stargateRouter cant be 0x0");
        ...
    }
```

Go back to Fix log

W1F: `totalSupply()` not guaranteed to be accurate

Impact:	Warning	Likelihood:	N/A
Target:	W1: `totalSupply()` not guaranteed to be accurate	Type:	Code logic

Description

The code logic has been adjusted so that the `totalSupply` is tracked accurately.

There is a new public variable `totalSupply` and the function `totalSupply()` has been removed.

The variable is:

- incremented in the `deposit()` function;
- decremented in the `withdraw()` function;
- decremented in `transferFrom()` to an auto-unwrap address.

Go back to Fix log

W2F: Renounce ownership

Impact:	Warning	Likelihood:	N/A
Target:	W2: Renounce ownership	Type:	Access controls

Description

The `renounceOwnership()` function has been overridden:

```
function renounceOwnership() public override {}
```

It is no longer possible to call this function accidentally and loose the ownership of the contract.

Go back to Fix log

I1F: Public functions

Impact:	Informational	Likelihood:	N/A
Target:	I1: Public functions	Type:	Gas optimization

Description

The following functions have been declared `external`, as was recommended:

- `withdraw()`
- `approve()`
- `transfer()`

There is no longer a function `totalSupply()` that could have been also declared external (see W1 fix). Therefore, all functions that could be declared external are declared external.

Go back to Fix log

I2F: Variables should be declared `constant`

Impact:	Informational	Likelihood:	N/A
Target:	I2: Variables should be declared `constant`	Type:	Gas optimization

Description

All variables discussed in I2 were declared `constant`, as was recommended.

Go back to Fix log

I3F: Unused imports

Impact:	Informational	Likelihood:	N/A
Target:	I3: Unused imports	Type:	Code quality

Description

All unused imports in RouterETH were removed from the code.

Go back to Fix log

I4F: Comments quality

Impact:	Informational	Likelihood:	N/A
Target:	I4: Comments quality	Type:	Code quality

Description

Both mistakes in the comments in StargateEthVault were fixed.

Go back to Fix log

I5F: Code quality

Impact:	Informational	Likelihood:	N/A
Target:	I5: Code quality	Type:	Code quality

Description

The interface `IStargateEthVault` is now declared in an external file and imported in both `StargateEthVault` and `RouterETH`. `StargateEthVault` implements the interface, as was recommended.

Go back to Fix log

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/wpM77gR7en