HALBORN

deBridge - Pre Release

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 18th, 2022 - February 9th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
1.0	Document Creation	02/09/2022	Timur Guvenkaya
1.1	Document Review	02/09/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Timur Guvenkaya	Halborn	Timur.Guvenkaya@halborn.com

EXECUTIVE OVERVIEW

1.1 Introduction

deBridge engaged Halborn to conduct a pre-release security assessment of the latest changes in their smart contracts. deBridge a cross-chain interoperability and liquidity transfer protocol that allows truly decentralized transfer of assets between various blockchains. The cross-chain intercommunication of deBridge smart contracts is powered by the network of independent oracles/validators, which are elected by deBridge governance.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure development.

1.2 Audit Summary

The team at Halborn assigned one full-time security engineer to audit the security of the assets in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security improvements that were acknowledged deBridge team.

1.3 Test Approach & Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to

the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the smart contract code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual testing of core functions through Hardhat and Ganache
- Manual testing with custom scripts.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Testnet deployment (Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 Scope

The review was scoped to the exact commit of 948ecacd510e93aa1c75881c213a34a52442ed10 of the contracts in the main branch of the debridge-contracts-v1 repository.

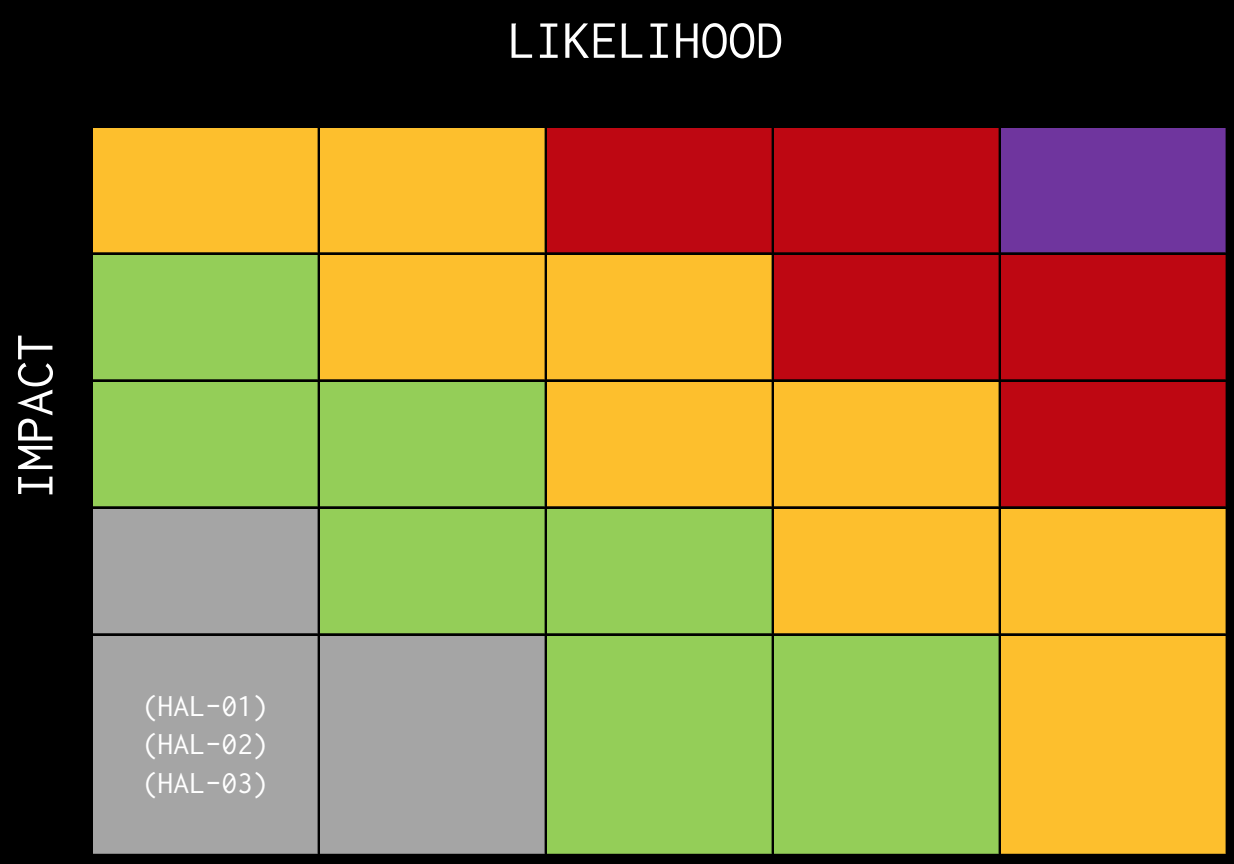
Smart contracts:

- DeBridgeGate.sol
- OraclesManager.sol
- DeBridgeTokenDeployer.sol
- WethGate.sol
- SignatureVerifier.sol
- CallProxy.sol
- DeBridgeTokenPaused.sol
- DeBridgeTokenProxy.sol
- DeBridgeToken.sol
- Claimer.sol
- SimpleFeeProxy.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3

EXECUTIVE OVERVIEW



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SOLC 0.8.7 COMPILER VERSION CONTAINS MULTIPLE BUGS	Informational	ACKNOWLEDGED
(HAL-02) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	ACKNOWLEDGED
(HAL-03) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) SOLC 0.8.7 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

Solidity compiler version 0.8.9 fixed important bugs in the compiler. The version 0.8.7 is missing some fixes:

- 0.8.9

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the most tested and stable versions, such as 0.6.12 or 0.7.6. Otherwise, if you still want to use ^0.8.0, because of the new functionality it provides, it is recommended to use 0.8.9 version.

Remediation Plan:

ACKNOWLEDGED: The deBridge team acknowledged this issue.

3.2 (HAL-02) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In all the loops, the variable `i` is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`.

Code Location:

Claimer.sol
- Line 70: `for (uint256 i = 0; i < claimsCount; i++)`
- Line 91: `for (uint256 i = 0; i < count; i++)`
- Line 115: `for (uint256 i = 0; i < count; i++)`
- Line 125: `for (uint256 i = 0; i < count; i++)`

DeBridgeToken.sol
- Line 71: `for (uint256 i = 0; i < mintersCount; i++)`

DeBridgeGate.sol
- Line 360: `for (uint256 i = 0; i < _chainIds.length; i++)`
- Line 394: `for (uint256 i = 0; i < _supportedChainIds.length; i++)`
- Line 554: `for (uint256 i = 0; i < _submissionIds.length; i++)`

DeBridgeTokenDeployer.sol
- Line 177: `for (uint256 i = 0; i < _debridgeIds.length; i++)`

OraclesManager.sol
- Line 85: `for (uint256 i = 0; i < _oracles.length; i++)`
- Line 125: `for (uint256 i = 0; i < oracleAddresses.length; i++)`

SignatureVerifier.sol
- Line 73: ‘for (uint256 i = 0; i < signaturesCount; i++)‘‘

Proof of Concept:

For example, based in the following test contract:

Listing 1: Test.sol

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8.9;
3
4 contract test {
5     function postiincrement(uint256 iterations) public {
6         for (uint256 i = 0; i < iterations; i++) {
7         }
8     }
9     function preiincrement(uint256 iterations) public {
10        for (uint256 i = 0; i < iterations; ++i) {
11        }
12    }
13 }
```

We can see the difference in the gas costs:

```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59balcf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59balcf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of an uint variable inside a loop. This is not applicable outside of loops.

Remediation Plan:

ACKNOWLEDGED: The deBridge team acknowledged this issue.

3.3 (HAL-03) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0 - INFORMATIONAL

Description:

`uint256` variables are already initialized to `0` by default. `uint256 i = 0`, for example, reassigns the `0` to `i` which wastes gas.

Code Location:

Claimer.sol
- Line 70: `for (uint256 i = 0; i < claimsCount; i++)`
- Line 91: `for (uint256 i = 0; i < count; i++)`
- Line 115: `for (uint256 i = 0; i < count; i++)`
- Line 125: `for (uint256 i = 0; i < count; i++)`

DeBridgeToken.sol
- Line 71: `for (uint256 i = 0; i < mintersCount; i++)`

DeBridgeGate.sol
- Line 360: `for (uint256 i = 0; i < _chainIds.length; i++)`
- Line 394: `for (uint256 i = 0; i < _supportedChainIds.length; i++)`
- Line 554: `for (uint256 i = 0; i < _submissionIds.length; i++)`

DeBridgeTokenDeployer.sol
- Line 177: `for (uint256 i = 0; i < _debridgeIds.length; i++)`

OraclesManager.sol
- Line 85: `for (uint256 i = 0; i < _oracles.length; i++)`
- Line 125: `for (uint256 i = 0; i < oracleAddresses.length; i++)`

SignatureVerifier.sol
- Line 73: ‘for (uint256 i = 0; i < signaturesCount; i++)‘‘

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not initialize uint256 variables to 0 to save some gas. For example: for (uint256 i; i < tokens.length; ++i){

Remediation Plan:

ACKNOWLEDGED: The deBridge team acknowledged this issue.

THANK YOU FOR CHOOSING

// HALBORN