// HALBORN

DRAFT

Spherium Finance - Bridge

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 17th, 2021 - August 21st, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/19/2021	Juned Ansari
0.9	Document Edits	08/19/2021	Juned Ansari
1.0	Final Review	08/20/2021	Gabi Urrutia
1.1	Remediation Plan	08/23/2021	Juned Ansari
1.1	Remediation Plan Review	08/26/2021	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
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Spherium Finance offers a complete suite of financial services comprising a universal wallet, token swap platform, money markets, and inter-blockchain liquidity transfer.

Spherium engaged Halborn to conduct a security assessment on their Bridge smart contracts beginning on August 17th, 2021 and ending August 21st, 2021. This security assessment was scoped to the Bridge smart contracts code in Solidity.

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 was provided a week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security 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 to achieve the following:

- Ensure that all bridge functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified several security risk which were addressed by Spherium Finance 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 this audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the bridge 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 assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- 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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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

IN-SCOPE : Spherium-Bridge Smart Contracts
The security assessment was scoped to the following smart contract:

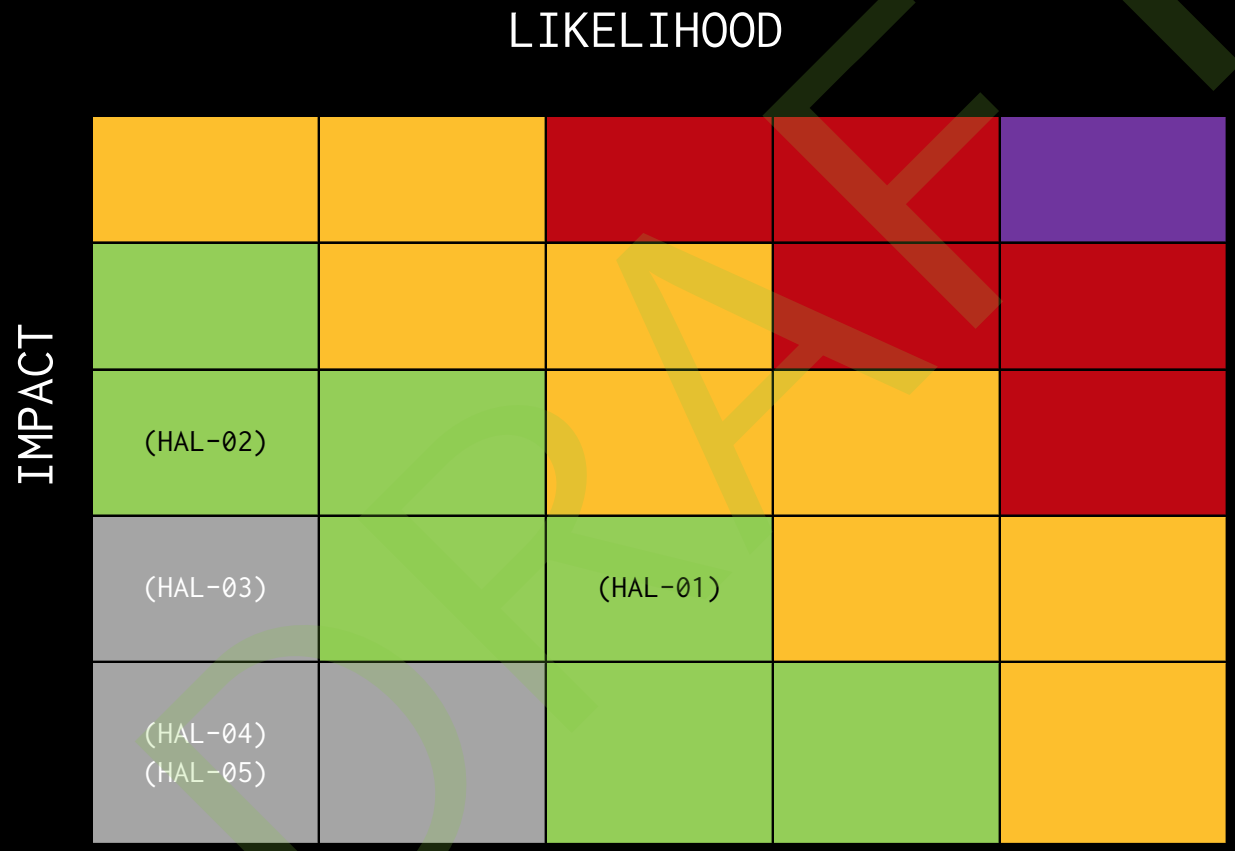
Listing 1: Bridge Contract

```
1 Bridge.sol
```

OUT-OF-SCOPE : External libraries and economics attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - IGNORE RETURN VALUES	Low	SOLVED - 08/21/2021
HAL02 - FLOATING PRAGMA	Low	SOLVED - 08/21/2021
HAL03 - PRAGMA VERSION	Informational	SOLVED - 08/21/2021
HAL04 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 08/21/2021
HAL05 - UNUSED CODE	Informational	SOLVED - 08/21/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In contract Bridge.sol, there is an instances where external methods are being called and return value(bool) is being ignored.

It was observed that Bridge.executeProposal(Bridge.Proposal) ignores return value by IERC20(proposal.tokenAddress).transfer(proposal.depositor,proposal.amount).

Code Location:

Listing 2: Bridge.sol (Lines 395,396,397,398)

```
388     function executeProposal(Proposal storage proposal) private
            whenNotPaused {
389         if (_burnList[proposal.tokenAddress]) {
390             IERC20(proposal.tokenAddress).mint(
391                 proposal.depositor,
392                 proposal.amount
393             );
394         } else {
395             IERC20(proposal.tokenAddress).transfer(
396                 proposal.depositor,
397                 proposal.amount
398             );
399         }
400     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendations:

Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

Remediation Plan:

SOLVED: Spherium.Finance Team updated the code and added the return value.

Listing 3: Bridge.sol (Lines 393,394,395,396,397)

```
392         } else {
393             bool result = IERC20(proposal.tokenAddress).transfer(
394                 proposal.depositor,
395                 proposal.amount
396             );
397             require(result, "unsuccessful transfer");
398         }
```

3.2 (HAL-02) FLOATING PRAGMA - LOW

Description:

Bridge Smart contract uses the floating pragma ^0.8.0. Contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code Location:

Listing 4: (Lines 1)

```
1 pragma solidity ^0.8.0;
2 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider locking the pragma version with known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: Spherium.Finance Team locked the pragma version.

3.3 (HAL-03) PRAGMA VERSION - INFORMATIONAL

Description:

Bridge contract uses one of the latest pragma version (0.8.0) which was released on December 16, 2020. The latest pragma version (0.8.7) was released in August 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

Reference: https://github.com/ethereum/solidity/releases

In the Solitidy Github repository, there is a json file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Code Location:

Listing 5: (Lines 1)

```
1 pragma solidity ^0.8.0;
2 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

Remediation Plan:

SOLVED: Spherium.Finance Team updated the code currently uses pragma version 0.7.6.

3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

Mark below functions as external instead of public:

Bridge.sol:
getDepositCount, addWhitelistToken, deposit, withdraw, withdrawBridgeFee

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practices, you should use external if you expect that the function will only ever be called externally and use public if you need to call the function internally. Mainly, Marking both function as external can save gas.

Remediation Plan:

SOLVED: Spherium.Finance Team updated the code and declared external functions instead of public.

3.5 (HAL-05) UNUSED CODE - INFORMATIONAL

Description:

During the test, It has been observed that some of the contract codes not used. There are a few instances of unused code (dead code) in the Bridge.sol.

Code Location:

Listing 6: (Lines 25)

```
25     uint8 chainID;
```

Listing 7: (Lines 334)

```
334 function withdraw() public {}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Unused codes should be deleted.

Remediation Plan:

SOLVED: Spherium.Finance Team removed the unused code in the contract.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

Bridge.sol

```
INFO:Detectors:
Reentrancy in Bridge.deposit(address,uint8,uint256) (contracts/Bridge.sol#285-332):
        External calls:
        - IERC20(tokenAddress).burnFrom(msg.sender,amount) (contracts/Bridge.sol#306)
        Event emitted after the call(s):
        - Deposit(destinationChainID,depositcount,amount,msg.sender,tokenAddress,DepositType.BURN) (contracts/Bridge.sol#308-315)
Reentrancy in Bridge.deposit(address,uint8,uint256) (contracts/Bridge.sol#285-332):
        External calls:
        - result = IERC20(tokenAddress).transferFrom(msg.sender,address(this),amount) (contracts/Bridge.sol#317-321)
        Event emitted after the call(s):
        - Deposit(destinationChainID,depositcount,transferAmount,msg.sender,tokenAddress,DepositType.LOCK) (contracts/Bridge.sol#323-330)
Reentrancy in Bridge.voteProposal(address,uint8,uint256,address,uint256) (contracts/Bridge.sol#334-384):
        External calls:
        - executeProposal(proposal) (contracts/Bridge.sol#372)
                - IERC20(proposal.tokenAddress).mint(proposal.depositor,proposal.amount) (contracts/Bridge.sol#388-391)
                - result = IERC20(proposal.tokenAddress).transfer(proposal.depositor,proposal.amount) (contracts/Bridge.sol#393-396)
        Event emitted after the call(s):
        - VoteProposal(tokenAddress,destinationChainID,depositcount,depositor,amount,proposal.status) (contracts/Bridge.sol#375-382)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Bridge.constructor(address[],uint256).i (contracts/Bridge.sol#125) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Bridge.constructor(address[],uint256) (contracts/Bridge.sol#116-129) has costly operations inside a loop:
        - relayersCount ++ (contracts/Bridge.sol#127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Bridge._onlyAdminOrRelayer() (contracts/Bridge.sol#131-137) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
removeWhitelistToken(address) should be declared external:
        - Bridge.removeWhitelistToken(address) (contracts/Bridge.sol#193-208)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

Bridge.sol

Report for contracts/Bridge.sol
https://dashboard.mythx.io/#/console/analyses/9ab1cbeb-82d4-441f-af8d-da2878df0734

Line	SWC Title	Severity	Short Description
25	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
26	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
63	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
357	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
360	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

All relevant valid findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN