// HALBORN

Parallel.fi

Bridge Solidity Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 24th, 2022 - February 14th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/14/2022	Timur Guvenkaya
0.2	Document Edits	02/15/2022	Hossam Mohamed
0.3	Document Edits	02/15/2022	Timur Guvenkaya
0.9	Document Edits	02/17/2022	Timur Guvenkaya
1.0	Remediation Plan	02/18/2022	Timur Guvenkaya
1.1	Remediation Plan Review	02/21/2022	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
Hossam Mohamed	Halborn	hossam.mohamed@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Parallel.fi engaged Halborn to conduct a security assessment on their Bridge Solidity smart contracts on January 24th, 2022 and ending February 14th, 2022. Parallel.fi is the decentralized platform that empowers everyone access to financial services built on Substrate.

1.2 AUDIT SUMMARY

The team at Halborn was provided 3 weeks for the engagement and assigned two full-time security engineers to audit the security of the assets in scope. The engineers are a blockchain and smart contract security experts 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 within the bridge solidity smart contracts

In summary, Halborn identified few security risks that should be addressed by Parallel.fi 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 solidity bridge smart contracts. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the 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. 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 solidity of smart contracts in the audit branch in parallel-token-bridge repository.

- Smart contracts
  - Bridge.sol
  - ERC20Safe.sol
  - Migrations.sol
  - ParallelBridgeToken.sol
  - AccessControl.sol
  - SafeCast.sol
  - SafeMath.sol
  - Pausable.sol
  - ERC20Handler.sol
  - HandlerHelpers.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	5	3	0

LIKELIHOOD

IMPACT

(HAL-06)		(HAL-01)		
		(HAL-02) (HAL-03)		
	(HAL-07)	(HAL-04) (HAL-05)		
		(HAL-08)		
		(HAL-09)		

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 FLAWED THRESHOLD/RELAYER DESIGN	High	SOLVED - 02/18/2022
HAL-02 IMPROPER ROLE BASED ACCESS CONTROL POLICY	Medium	SOLVED - 02/18/2022
HAL-03 ALL RELAYERS CAN BE REMOVED	Medium	SOLVED - 02/18/2022
HAL-04 OWNER CAN BE SET AS RELAYER	Medium	SOLVED - 02/18/2022
HAL-05 RELAYER CAN BE SET AS OWNER	Medium	SOLVED - 02/18/2022
HAL-06 ERC20SAFE.SAFECALL DOES NOT VERIFY THAT THE TOKEN ADDRESS IS A CONTRACT	Medium	SOLVED - 02/28/2022
HAL-07 MISSING ZERO ADDRESS CHECK	Low	SOLVED - 02/18/2022
HAL-08 MISSING ARRAY LENGTH CHECK	Low	SOLVED - 02/18/2022
HAL-09 MISSING EVENT EMITTING	Low	SOLVED - 02/18/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) HAL-01 FLAWED THRESHOLD/RELAYER DESIGN - HIGH

Description:

It was observed that the current relayer/threshold management design is not dependent on the numbers of relayers. Therefore, it is possible to have a threshold count much lower or higher than the relayer count.

Threshold of less than %80 of the total number of relayers:
- Increases the risk of malicious voting

Threshold of more than total relayer count:
- Inability to vote

Threshold equal to relayer count:
- If a relayer goes down, voting is not possible

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to make the voting threshold dependent on the number of relayers. In the current implementation, it is feasible to set the threshold to 80% of total relayers and update it each time a new relayer is added or removed.

Example Implementation:

Consider removing the adminChangeRelayerThreshold function and replacing it with the private function update_relayer_threshold which sets the voting threshold to 80% of total relayers. That function can be called every time a new relayer is added or removed

update_relayer_threshold

```
/**
        @notice Modifies the number of votes required for a proposal to be considered passed.

    */
function updateRelayerThreshold() private {
    uint256 new_threshold = (_totalRelayers() / 5) * 4; // %80 of total relayers
    _relayerThreshold = new_threshold.toUint8();
}
```

Calling in add/remove relayer functions

```
ftrace | funcSig
function adminAddRelayer(address relayerAddress⬆) external {
    require(
        relayerAddress⬆ ≠ address(0),
        "relayerAddress is the zero address"
    );
    require(
        !hasRole(RELAYER_ROLE, relayerAddress⬆),
        "addr already has relayer role!"
    );
    require(
        !hasRole(DEFAULT_ADMIN_ROLE, relayerAddress⬆),
        "addr has admin role!"
    );
    require(!hasRole(PAUSER, relayerAddress⬆), "addr has pauser role!");
    require(_totalRelayers() < MAX_RELAYERS, "relayers limit reached");
    grantRole(RELAYER_ROLE, relayerAddress⬆);

    updateRelayerThreshold();

    emit RelayerAdded(relayerAddress⬆);
}

/**
    @notice Removes relayer role for {relayerAddress}.
    @notice Only callable by an address that currently has the admin role, which is
            checked in revokeRole().
    @param relayerAddress Address of relayer to be removed.
    @notice Emits {RelayerRemoved} event.
 */
ftrace | funcSig
function adminRemoveRelayer(address relayerAddress⬆) external {
    require(
        hasRole(RELAYER_ROLE, relayerAddress⬆),
        "addr doesn't have relayer role!"
    );
    require(_totalRelayers() > MIN_RELAYERS, "relayers limit reached");
    revokeRole(RELAYER_ROLE, relayerAddress⬆);

    updateRelayerThreshold();
    emit RelayerRemoved(relayerAddress⬆);
}
```

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by following the suggested implementation.

3.2 (HAL-02) HAL-02 IMPROPER ROLE BASED ACCESS CONTROL POLICY - MEDIUM

Description:

It was observed that most of the privileged functionality is controlled by the owner. Additional authorization levels are needed to implement the principle of least privilege, also known as least authority, which ensures that only authorized processes, users, or programs can access the necessary resources or information. The ownership role is useful in a simple system, but more complex projects require more roles by using role-based access control. Although Bridge.sol has an additional Relayer role, there should be a separate role responsible for pausing/unpausing a smart contract if the owner is compromised.

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to add another role of Pauser to comply with the principle of least privilege and limit the privileges of owner.

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by adding the Pauser role.

3.3 (HAL-03) HAL-03 ALL RELAYERS CAN BE REMOVED - MEDIUM

Description:

It was observed that all relayers can be removed via the adminRemoveRelayer function.

Code Location:

Listing 1: Bridge.sol (Line 204)

```
204     function adminRemoveRelayer(address relayerAddress) external {
205         require(hasRole(RELAYER_ROLE, relayerAddress), "addr doesn
                't have relayer role!");
206         revokeRole(RELAYER_ROLE, relayerAddress);
207         emit RelayerRemoved(relayerAddress);
208     }
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to have a constant with minimum number of relayers and check with that

Example Implementation:

Listing 2: Bridge.sol (Line 205)

```
204 function adminRemoveRelayer(address relayerAddress) external {
205         require(hasRole(RELAYER_ROLE, relayerAddress), "addr doesn
                't have relayer role!");
206         require(_totalRelayers() > MIN_RELAYERS, "relayers limit
                reached");
207         revokeRole(RELAYER_ROLE, relayerAddress);
208         emit RelayerRemoved(relayerAddress);
209     }
```

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by adding the MIN_RELAYERS constant.

3.4 (HAL-04) HAL-04 OWNER CAN BE SET AS RELAYER - MEDIUM

Description:

It was observed that a contract Owner may also have a Relayer role. This situation violates the principle of least privilege, thus putting the contract at risk of potential role abuse.

Code Location:

- Owner can be set as Relayer in constructor function

Listing 3: Bridge.sol (Line 113)

```
113     constructor (uint8 domainID, address[] memory initialRelayers,
            uint256 initialRelayerThreshold, uint256 fee, uint256 expiry
            ) public {
114         _domainID = domainID;
115         _relayerThreshold = initialRelayerThreshold.toUint8();
116         _fee = fee.toUint128();
117         _expiry = expiry.toUint40();
118
119         _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
120
121         for (uint256 i; i < initialRelayers.length; i++) {
122             grantRole(RELAYER_ROLE, initialRelayers[i]);
123         }
124     }
```

- Owner can be set as Relayer in adminAddRelayer function

Listing 4: Bridge.sol (Line 190)

```
190     function adminAddRelayer(address relayerAddress) external {
191         require(!hasRole(RELAYER_ROLE, relayerAddress), "addr
                already has relayer role!");
192         require(_totalRelayers() < MAX_RELAYERS, "relayers limit
                reached");
193         grantRole(RELAYER_ROLE, relayerAddress);
194         emit RelayerAdded(relayerAddress);
195     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is not recommended to let the Owner also have the Relayer role. Please add a check that ensures that relayerAddress is not equal to the Owner address.

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by checking if the provided relayerAddress address is not equal to the owner address.

3.5 (HAL-05) HAL-05 RELAYER CAN BE SET AS OWNER - MEDIUM

Description:

It was observed that it is possible to set a Relayer to be also the contract owner. This situation violates the principle of least privilege, thus putting the contract at risk of potential role abuse.

Code Location:

- Relayer can be set as owner in renounceAdmin function

Listing 5: Bridge.sol (Line 172)

```
172  function renounceAdmin(address newAdmin) external onlyAdmin {
173          require(msg.sender != newAdmin, 'Cannot renounce oneself')
                 ;
174          require(newAdmin != address(0), "newAdmin is the zero
                 address");
175          grantRole(DEFAULT_ADMIN_ROLE, newAdmin);
176          renounceRole(DEFAULT_ADMIN_ROLE, msg.sender);
177      }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to add a check to ensures that newAdmin does not have a relayer role.

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by checking if the provided newAdmin address does not have a relayer role.

3.6 (HAL-06) ERC20SAFE.SAFECALL DOES NOT VERIFY THAT THE TOKEN ADDRESS IS A CONTRACT - MEDIUM

Description:

The _safeCall() function is used in the ERC20Safe contract to perform all token transfers:

Listing 6: ERC20Safe.sol

```
87   function _safeCall(IERC20 token, bytes memory data) private {
88           (bool success, bytes memory returndata) = address(token).
                 call(data);
89           require(success, "ERC20: call failed");
90
91           if (returndata.length > 0) {
92
93               require(abi.decode(returndata, (bool)), "ERC20:
                     operation did not succeed");
94           }
95       }
```

This function does not verify that the token address passed as a parameter is actually a contract that allows, as you can see below, to perform a transfer using zero address as a token parameter.

This was recently exploited in the Qubit Finance's bridge.

The likelihood of this exploit is very low since the token used in deposits must pass this check:
require(_contractWhitelist[tokenAddress], "provided tokenAddress is not whitelisted");

This scenario would only become real if an admin called Bridge.adminSetResource with an invalid/wrong tokenAddress.

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

It is recommended to follow OpenZeppelin approach and check in the _safeCall() function that the token address is a contract address. To achieve that, the function functionCall() instead of call() from OpenZeppelin Address.sol contract can be used.

Furthermore, another possible addition would be to check the balance before and after the asset transfer to ensure that the number of the transferred assets compiles the expectation.

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by using the isContract() check:

Listing 7: ERC20Safe.sol (Line 90)

```
89     function _safeCall(IERC20 token, bytes memory data) private {
90         require(address(token).isContract(), "Address: call to non
               -contract");
91         (bool success, bytes memory returndata) = address(token).
               call(data);
92         require(success, "ERC20: call failed");
93
94         if (returndata.length > 0) {
95             require(abi.decode(returndata, (bool)), "ERC20:
                   operation did not succeed");
96         }
97     }
98 }
```

3.7 (HAL-07) HAL-06 MISSING ZERO ADDRESS CHECK - LOW

Description:

Lack of zero address check was observed in Bridge.sol. Each address must be validated and checked that it is not a zero address. This is also considered a best practice.

Code Location:

Listing 8: Bridge.sol (Lines 98,102,141,150,190,240,218,230,286)

```
1 function _relayerBit()
2
3 function _hasVoted()
4
5 function isRelayer()
6
7 function renounceAdmin()
8
9 function adminAddRelayer()
10
11 function adminRemoveRelayer()
12
13 function adminSetResource()
14
15 function adminSetBurnable()
16
17 function adminWithdraw()
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to validate that each address input is not a zero address.

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by adding all zero address checks.

3.8 (HAL-08) HAL-07 MISSING ARRAY LENGTH CHECK - LOW

Description:

It was observed that in Bridge.sol, the transferFunds function lacks array length checking on passed arguments. If addr.length != amounts.length, the transaction will fail.

Code Location:

Listing 9: Bridge.sol (Line 453)

```
452     function transferFunds(address payable[] calldata addrs, uint
            [] calldata amounts) external onlyAdmin {
453         for (uint256 i = 0; i < addrs.length; i++) {
454             addrs[i].transfer(amounts[i]);
455         }
456     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Add check to ensure that addr.length == amounts.length
Example Code:

Listing 10: Bridge.sol

```
452 function transferFunds(address payable[] calldata addrs, uint[]
      calldata amounts) external onlyAdmin {
453         require(address.length == amounts.length, "Length of
              addresses and amounts must match");
454
455         for (uint256 i = 0; i < addrs.length; i++) {
456             addrs[i].transfer(amounts[i]);
457         }
458     }
```

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by adding the array length check.

3.9 (HAL-09) HAL-08 MISSING EVENT EMITTING - LOW

Description:

It has been observed that important functionality is missing emitting event for a function in the Bridge.sol contract. Functions must emit events. Events are a method of informing the transaction initiator about the actions performed by the called function. It logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters. These functions should emit events.

Code Location:

Listing 11: Bridge.sol (Lines 218,230,274,286)

```
1 function adminSetResource()
2
3 function adminSetBurnable()
4
5 function adminChangeFee()
6
7 function adminWithdraw()
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

For best security practices, consider as much as possible, declaring events at the end of the function. Events can be used to detect the end of the operation.

Remediation Plan:

SOLVED: The Parallel.fi team has solved the issue by adding all the relevant events.

AUTOMATED TESTING

4.1 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits 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 all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

- No issues found by MythX

THANK YOU FOR CHOOSING

// HALBORN