RSK Federated Bridge

Security Assessment

August 3, 2020

Prepared For:
Pedro Prete | *RSK*
pedro@iovlabs.org

Maximiliano Del Hoyo | *RSK*
mdh@rsk.co

Adrian Eidelman | *RSK*
adrian@iovlabs.org

Prepared By:
Will Song | *Trail of Bits*
will.song@trailofbits.com

Sam Sun | *Trail of Bits*
sam.sun@trailofbits.com

Executive Summary

From June 27 through June 31, 2020, RSK engaged Trail of Bits to review the security of the Ethereum-RSK Federated Bridge smart contracts. Trail of Bits conducted this assessment over the course of one person-week with two engineers working from `d4e55708a7910bd2f14b511b1fe5d9fc1888272a` from the `rsksmart/tokenbridge` repository.

The smart contracts were reviewed for implementation correctness and security issues. We also reviewed the resolutions from the Coinspect draft report and discovered a total of five findings:

- Three findings are informational severity and do not pose any immediate risk.
- One low-severity finding allows the contract deployer to violate constraints on the maximum number of federation members imposed elsewhere in the `Federation_v1` smart contract.
- One medium-severity finding allows the contract owner to violate the constraint on the minimum number of federation members.

We found the smart contract code to be very clear and concise, with a well-defined purpose. The usage of Solidity modifiers for preconditions is to be appreciated. The smart contracts are accompanied by a comprehensive `Truffle.js` test suite which tests many of the interesting code paths in the contracts. However, the low-severity and medium-severity findings stem from brief lapses in judgement, which could be discovered automatically via property testing.

Our five short-term recommendations are immediately actionable and do not require much effort. We urge RSK to prioritize addressing the two non-informational findings related to validating `members.length` in the `Federation_v1` contract—these can be resolved with additional Solidity modifiers.

On August 18, Trail of Bits reviewed the resolutions merged into the `rsksmart/tokenbridge` repository, using the commit hash `28d1992f18b321490d616ab9ff135df950285bcb`. We conclude that all short-term recommendations aside from TOB-RSK-001 have been correctly implemented.

Project Dashboard

Application Summary

Name	RSK Federated Bridge
Version	`d4e55708a7910bd2f14b511b1fe5d9fc1888272a (review)` `28d1992f18b321490d616ab9ff135df950285bcb (retest)`
Type	Whitebox
Platforms	Solidity, EVM

Engagement Summary

Dates	July 27– July 31
Method	Whitebox
Consultants Engaged	2
Level of Effort	1 person-week

Vulnerability Summary

Total High-Severity Issues	0	
Total Medium-Severity Issues	1	■
Total Low-Severity Issues	1	■
Total Informational-Severity Issues	3	■ ■ ■
Total Undetermined-Severity Issues	0	
Total	5	

Category Breakdown

Data Validation	3	■ ■ ■
Configuration	1	■
Auditing and Logging	1	■
Total	5	

Code Maturity Evaluation

Here we review the maturity of the codebase and the likelihood of future issues. In each area of control, we rate the maturity from strong to weak, or missing, and give a brief explanation of our reasoning.

Category Name	**Description**
Access Controls	**Moderate.** We report two issues that fail to correctly validate `members.length`.
Arithmetic	**Satisfactory.** We report one typing issue which could lead to future bugs.
Assembly Use	**Not Reviewed.**
Centralization	**Not Reviewed.**
Upgradeability	**Not Reviewed.**
Function Composition	**Strong.** We found the codebase to be separated into functions with clear purpose. We have no concerns in this area.
Front-Running	**Not Reviewed.**
Key Management	**Not Reviewed.**
Monitoring	**Not Reviewed.**
Testing & Verification	**Strong.** We note that the smart contracts have an extensive `Truffle.js` test suite and the testing-related finding from Coinspect has been resolved.

Engagement Goals

The engagement was scoped to provide a security assessment of the Ethereum smart contracts for the RSK Federated Bridge. When `v0` and `v1` contracts were both present, we focused our efforts on the `v1` contracts, following a brief discussion in the shared Slack channel.

Specifically, we sought to answer the following questions:

- Do the smart contracts contain any common Solidity bugs, such as re-entrancy?
- To the best of our knowledge, are the smart contracts implemented correctly?
- Does PR #122 correctly implement our recommendations?

Coverage

Smart Contracts. Our extensive manual review of the Solidity code revealed five findings. Additionally, we ran the `crytic.io` analyzer on the smart contract subdirectory, which did not uncover any findings.

Retesting. A brief code review was conducted on the diff report corresponding to GitHub pull request #122. We discovered a faulty fix for TOB-RSK-001 and verified that the other changes are correct.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

☐ **Validate `members.length` against `MAX_MEMBER_COUNT` during contract deployment.** This ensures that the members array satisfies the upper bound. TOB-RSK-001

☐ **Validate the new member count with the requirement when removing members.** This ensures that the members array satisfies the lower bound. TOB-RSK-002

☐ **Change the loop variable type to `uint`** to avoid an infinite loop when `MAX_MEMBER_COUNT` is raised above 255. TOB-RSK-003

☐ **Make the Utils contract a library.** The Utils contract has no state so it does not need to be a contract. TOB-RSK-004

☐ **Emit an event when the bridge is changed.** This owner-restricted action does not emit an event. TOB-RSK-005

Long Term

☐ **Review all stateful actions and ensure that enough logs are emitted.** TOB-RSK-005

Findings Summary

#	Title	Type	Severity
1	Max member limit can be bypassed at deploy time	Data Validation	Low
2	Too many members may be removed, violating the minimum member requirement	Data Validation	Medium
3	`MAX_MEMBER_COUNT` and loop variable have different types	Data Validation	Informational
4	`Utils.sol` can be a library	Configuration	Informational
5	Setting a new bridge does not emit an event	Auditing and Logging	Informational

1. Max member limit can be bypassed at deploy time

Severity: Low
Difficulty: Low
Type: Data Validation
Finding ID: TOB-RSK-001
Target: `Federation_v1.sol`

Description

The `MAX_MEMBER_COUNT` constant limits the maximum number of members that can be active in the federation at once. During contract construction, the number of members is not checked against this constant, and allows for federation sizes that are larger than expected.

```
constructor(address[] memory _members, uint _required)
    public validRequirement(_members.length, _required) {
    members = _members;
    for (uint i = 0; i < _members.length; i++) {
        require(!isMember[_members[i]] && _members[i] != NULL_ADDRESS,
                "Federation: Invalid members");
        isMember[_members[i]] = true;
        emit MemberAddition(_members[i]);
    }
    required = _required;
    emit RequirementChange(required);
}
```

Figure 1.1: The `Federation_v1` *constructor.*

Exploit Scenario

The contract deployer deploys the contract with the members array initialized to 100 different addresses.

Recommendation

Validate `members.length` against `MAX_MEMBER_COUNT` during contract deployment.

Resolution

`_members.length < MAX_MEMBER_COUNT` is implemented in commit `28d1992f18b321490d616ab9ff135df950285bcb`, which results in an off-by-one error in which only 49 members can be constructed, with a maximum of 50. The last member can be added via `addMember`.

2. Too many members may be removed, violating the minimum member requirement

Severity: Medium
Type: Data Validation
Target: `Federation_v1.sol`

Difficulty: Low
Finding ID: TOB-RSK-002

Description
There is no check to validate that the number of members meets at least the minimum requirement when members are removed from the federation.

```
function removeMember(address _oldMember) external onlyOwner
{
    require(_oldMember != NULL_ADDRESS, "Federation: Empty member");
    require(isMember[_oldMember], "Federation: Member doesn't exists");
    require(members.length > 1, "Federation: Can't remove all the members");

    isMember[_oldMember] = false;
    for (uint i = 0; i < members.length - 1; i++) {
        if (members[i] == _oldMember) {
            members[i] = members[members.length - 1];
            break;
        }
    }
    members.length -= 1;
    emit MemberRemoval(_oldMember);
}
```

Figure 2.1: `removeMember` *and its requirements.*

Exploit Scenario
A malicious minority manages to convince the owner to remove more members than allowed by the minimum requirement so that the malicious minority will become the majority.

Recommendation
Validate the new member count with the requirement when removing members.

Resolution
A `require` asserting that the new member count is at least the minimum number required has been added to commit `28d1992f18b321490d616ab9ff135df950285bcb`.

3. MAX_MEMBER_COUNT and loop variable have different types

Severity: Informational
Difficulty: Low
Type: Data Validation
Finding ID: TOB-RSK-003
Target: `Federation_v1.sol`

Description

The loop variable of `getTransactionCount` has a smaller type than the `MAX_MEMBER_COUNT` upper bound. This could increase the chance of a future bug if `MAX_MEMBER_COUNT` is ever raised beyond 255.

```
function getTransactionCount(bytes32 transactionId) public view returns(uint8) {
    uint8 count = 0;
    for (uint8 i = 0; i < members.length; i++) {
        if (votes[transactionId][members[i]])
            count += 1;
    }
    return count;
}
```

Figure 3.1: `getTransactionCount` *loop variable.*

Recommendation

Change the type of the loop variable to `uint`.

Resolution

Commit `28d1992f18b321490d616ab9ff135df950285bcb` changes both the loop variable and the return type of `getTransactionCount` to `uint`.

4. `Utils.sol` can be a library

Severity: Informational
Difficulty: Low
Type: Configuration
Finding ID: TOB-RSK-004
Target: `Utils.sol`

Description
The utility smart contract presents an interface without any stateful changes, so it would make much more sense to call into it as a library rather than as a smart contract.

Recommendation
Make the Utils contract a library.

Resolution
Commit `28d1992f18b321490d616ab9ff135df950285bcb` changes this contract to a library.

5. Setting a new bridge does not emit an event

Severity: Informational
Type: Auditing and Logging
Target: `Federation_v1.sol`

Difficulty: Low
Finding ID: TOB-RSK-005

Description

While all other owner-restricted actions emit an event for auditing purposes, the function `setBridge` does not, as shown in Figure 5.1. This makes it difficult for a third party to audit the history of the bridge used by the federation.

```
function setBridge(address _bridge) external onlyOwner {
    require(_bridge != NULL_ADDRESS, "Federation: Empty bridge");
    bridge = IBridge_v1(_bridge);
}
```

Figure 5.1: Lack of event in `setBridge`*.*

Recommendation

Short term, emit an event when the bridge is changed.

Long term, review all functions and ensure that enough logs are emitted such that an off-chain user is able to determine what state changes occurred in the contract even if the function was called in an internal transaction.

Resolution

The new event `BridgeChanged` has been created and emitted in commit `28d1992f18b321490d616ab9ff135df950285bcb`.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for

	client’s reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client’s reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes

Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria

Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.