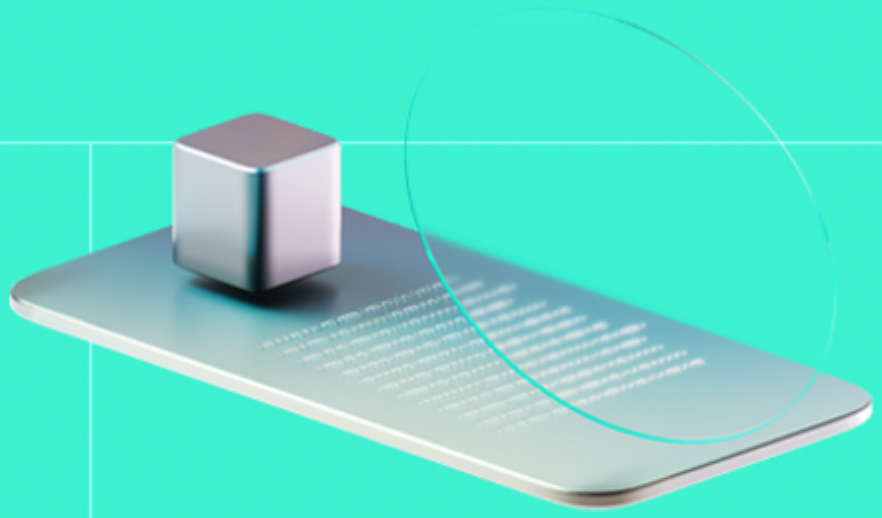
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Aviator

Date: 05/04/2024

We express our gratitude to the Aviator team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

SkyBridge by Aviator Technologies, LLC is a custom bridge and launchpad solution between the Ethereum Layer-1 and the Base Layer-2 networks that supports any arbitrary ERC-20 token.

Platform: EVM, Optimism

Language: Solidity

Tags: Bridge, Liquidity Pool.

Timeline: 08/03/2024 - 05/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/AviatorAC/skybridge
Commit	33a578d7766781cb1366355797b0c20759ffd856

Audit Summary

10/10	10/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

3	3	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	1
Medium	1
Low	1

Vulnerability	Status
F-2024-1381 - Unrestricted Fees	Fixed
F-2024-1500 - Mismatch Between Documentation and Implementation	Fixed
F-2024-1511 - Unexpected Allowance Required to Bridge Tokens	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Aviator
Audited By	Maksym Fedorenko, Roman Tiutiun
Approved By	Grzegorz Trawinski
Website	https://hacken.io
Changelog	15/03/2024 - Preliminary Report, 05/04/2024 - Final Report

Table of Contents

System Overview

SkyBridge is a custom bridge and launchpad solution between the Ethereum Layer-1 and the Base Layer-2 networks that supports any arbitrary ERC-20 token. with the following contracts:

- L1AviERC721Bridge.sol — is the bridging of ERC721 tokens between different domains (such as Ethereum Layer 1 and Layer 2 networks).
- L1Bridge.sol — is responsible for transferring ETH and ERC20 tokens between L1 and L2. In the case that an ERC20 token is native to L1, it will be escrowed within this contract.
- LiquidityPool.sol — is utilized to provide a central place to hold funds for fast bridging and the fees from Layer 1 to Layer 2 transfers.
- L2AviERC721Bridge.sol — is a contract that works together with the L2 ERC721 bridge to make it possible to transfer ERC721 tokens from Ethereum to Optimism. This contract acts as an escrow for ERC721 tokens deposited into L2.
- L2Bridge.sol — is responsible for transferring ETH and ERC20 tokens between L1 and L2. In the case that an ERC20 token is native to L2, it will be escrowed within this contract. If the ERC20 token is native to L1, it will be burnt.
- AviPredeploys.sol — it contains constants representing predeployed contract addresses on Layer 1 (L1) and Layer 2 (L2) networks.
- AviBridge.sol — is a base contract for the L1 and L2 standard ERC20 bridges. It handles the core bridging logic, including escrowing tokens that are native to the local chain and minting/burning tokens that are native to the remote chain.
- AviERC721Bridge.sol — it serves as a base contract for both the Layer 1 (L1) and Layer 2 (L2) ERC721 bridges within the Avi ecosystem.

Privileged roles

- Bridge Admin: Full control over the bridges. Able to change the fee structure, change the address of the liquidity pool, and other bridges, commit the fast withdrawals using the liquidity from the liquidity pool.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical requirements are provided.

Code quality

The total Code Quality score is **10** out of **10**.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- The contracts and libraries are tested thoroughly.

Security score

Upon auditing, the code was found to contain **0** critical, **1** high, **1** medium, and **1** low severity issues, leading to a security score of **4** out of **10**. Upon the retest, all issues were fixed resulting in the final score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The admin can withdraw all the liquidity form the liquidity pool.
- The admin can increase the bridging fees up to 10% and flat fees up to 0.005 ETH.
- The admin can prevent the bridging event transfer by updating the bridge configuration.
- The admin can pause the system.

Findings

Vulnerability Details

F-2024-1500 - Mismatch Between Documentation and Implementation - High

Description:

The documentation for the **Fee Structure** specifies that:

> 0.3% of tokens bridged are harvested as bridging fees and added to the bridge liquidity to facilitate fast transfers;
>
> The native protocol token, Aviator (AVI), is excluded from the 0.3% liquidity fee.
>
> 0.002 Ethereum is the protocol fee for bridging ERC-721 tokens.

However, upon reviewing the actual implementation of the `L1AviBridge.sol` and `L1AviERC721Bridge.sol` contract, it is evident that there is `2.5%` fees for all the tokens including the `AVI` and no fees for ERC721 tokens.

This discrepancy between the documentation and the contract's code leads to confusion and potential misunderstandings about the contract's behavior and capabilities.

Assets:

- L1/L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1/L1Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Severity: High

Impact:

Impact [1-5]: 4
Exploitability [1-2]: 1
Complexity [0-2]: 1
Final Score: 4.3 (High)

Recommendations

Recommendation: Review the documentation and update the implementation to match the expected result.

Remediation (Revised commit: 4e50fe): The code and documentation were updated to:
ERC20:

- 0.001 Ethereum is the protocol fee.
- 0.3% of tokens bridged are harvested as bridging fees and added to the bridge liquidity to facilitate fast transfers. (Ex: $1000 USDC = $3 USDC liquidity fee).

ERC721:

- 0.002 Ethereum is the protocol fee.

F-2024-1511 - Unexpected Allowance Required to Bridge Tokens - Medium

Description:

The contract `L1AviBridge` has the function `_initiateERC20Deposit` which is responsible for initiating bridging of ERC20 tokens. The function accept the amount of tokens `_amount` which should be transferred to another chain, calculates the fees `fee` and transfers the fees to the liquidity pool `LIQUIDITY_POOL`, and locks the tokens without subtracting the amount transferred for the fees.

This might leads to the user confusion as the amount of tokens which should be approved before calling the function is different from the amount utilized by the function, also it contradicts the `_initiateETHDeposit` function implementation where the `_amount` includes the fees.

Assets:

- L1/L1Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 4
Impact [1-5]: 2
Exploitability [1-2]: 1
Complexity [0-2]: 0
Final Score: 3 (Medium)

Recommendations

Recommendation:

It is recommended to rework the logic in order to subtract the fees from the amount which is intended to be transferred.

Remediation (Revised commit: 57e8060): The code was updated to ensure the required allowance matches the value specified by the `_amount` argument in the `_initiateERC20Deposit` function.

F-2024-1381 - Unrestricted Fees - Low

Description:

The `L1AviBridge.sol` contract currently does not enforce any limitations on the system fees that can be applied, allowing for the possibility of any fees, including 100% or more. `setFlatFee()`, `setBridgingFee()` functions do not impose any restrictions.

```
function setFlatFee(uint256 _fee) external onlyRole(DEFAULT_ADMIN_ROLE) {
flatFee = _fee;
}

function setBridgingFee(uint256 _fee) external onlyRole(DEFAULT_ADMIN_ROLE) {
bridgingFee = _fee;
}
```

This might lead to the denial of service if a value higher than 100% is specified or to the users' unexpected expenses if the fee is updated after the users' deposit is made.

Assets:

- L1/L1Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Severity: Low

Recommendations

Recommendation: Add the validation to ensure that the fees are lower than the threshold, for example, 20%.

Remediation (Revised Commit: 5b701ee): The pragma version is set to `0.8.15` in all contracts.

Observation Details

F-2024-1367 - Floating Pragma - Info

Description: A `LiquidityPool.sol` uses floating pragmas `^0.8.15` version.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- L1/LiquidityPool.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Recommendation: Consider locking the pragma version in all contracts.

Remediation (Revised Commit: 5b701ee): The pragma version is set to `0.8.15` in all contracts.

F-2024-1378 - Missing Zero Address Validation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, consider a contract that includes a function to change its owner. This function is crucial, as it determines who has administrative access. However, if this function lacks proper validation checks, it might inadvertently permit the setting of the owner to the zero address. Consequently, the administrative functions will become unusable.

Function `sendETH()` in `LiquidityPool.sol`, `constructor()` in `L1AviBridge.sol`, `setOtherBridge()` in `L1AviBridge.sol`, `constructor()` in `L2AviBridge.sol` are lack of missing zero address validation.

Assets:

- L1/L1Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1/LiquidityPool.sol [https://github.com/AviatorAC/skybridge]
- L2/L2Bridge.sol [https://github.com/AviatorAC/skybridge]
- iversal/AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Recommendation:

Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Remediation (Revised Commit: 5b701ee): The zero address validation for function `sendETH()` in `LiquidityPool.sol`, `constructor()` in `L1AviBridge.sol`, `setOtherBridge()` in `L1AviBridge.sol`, `constructor()` in `L2AviBridge.sol`, were implemented.

F-2024-1486 - Precense of Redundant Code From The Upgradable Implementation - Info

Description: The system of smart contracts is not upgradable, however it has redundant code which is common for the upgradable smart contracts, such proxy implementation `Initializer` by Openzeppelin, presence of `__gap` state array.

This introduce the code redundancy, decreases readability and results in Gas expenses.

Assets:

- L1/L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1/L1Bridge.sol [https://github.com/AviatorAC/skybridge]
- L2/L2AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L2/L2Bridge.sol [https://github.com/AviatorAC/skybridge]
- iversal/AviBridge.sol [https://github.com/AviatorAC/skybridge]
- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Recommendation: Remove the code related to the upgradable implementation, move the code from the Initializer to the constructor.

Remediation (Revised Commit: 5b701ee): The issue was fixed by moving the `Initializer` to the `constructor`, and code related to the upgradable implementation was removed.

F-2024-1490 - Redundant Getters For The Public State Variables - Info

Description: The contract has public state variables `MESSENGER`, and `OTHER_BRIDGE`, but also it has getters functions, such as `messenger()` and `otherBridge()`. This introduces code redundancy, during the compilation the getter is created for each public variable.

```
AviBridge public OTHER_BRIDGE;

function otherBridge() external view returns (AviBridge) {
return OTHER_BRIDGE;
}
```

This increases the deployment Gas cost.

Assets:

- iversal/AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Recommendation: Set public variables to private, internal or remove the getters.

Remediation (Revised Commit: 5b701ee): The recommendations were implemented by adding visibility `internal` to the variables.

F-2024-1505 - Non-Compliance with Naming Conventions for Contract and File Names - Info

Description: Solidity best practices suggest that contract names should match their file names and both should be in PascalCase. PascalCase (or Upper Camel Case) is a naming convention where each word begins with a capital letter without spaces. For example, `L1AviBridge`.

The observed inconsistency in the contract codebase can lead to confusion and reduce code readability and maintainability.

Assets:

- L1/L1Bridge.sol [https://github.com/AviatorAC/skybridge]
- L2/L2Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Recommendation: It is suggested to modify the file name accordingly to the contract's name the file is hosting.

- current format: `L1Bridge.sol`, `L2Bridge.sol`
- suggested format: `L1AviBridge.sol`, `L2AviBridge.sol`

Remediation (Revised Commit: 5b701ee): The issue was fixed by renaming the contract to `L1AviBridge.sol`, `L2AviBridge.sol`.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Primary Scope

Details

Repository	https://github.com/AviatorAC/skybridge
Commit	33a578d7766781cb1366355797b0c20759ffd856
Whitepaper	./Aviator%20SkyBridge%20Technical%20Documenation%20v1.0.pdf
Requirements	./Aviator%20SkyBridge%20Technical%20Documenation%20v1.0.pdf
Technical Requirements	./Aviator%20SkyBridge%20Technical%20Documenation%20v1.0.pdf

Secondary Scope

Details

Repository	https://github.com/AviatorAC/skybridge
Commit	4e50feca34b10d1c0181699f2f4f114870bb29ba
Whitepaper	./Aviator%20SkyBridge%20Technical%20Documenation%20v1.01.pdf
Requirements	./Aviator%20SkyBridge%20Technical%20Documenation%20v1.01.pdf
Technical Requirements	./Aviator%20SkyBridge%20Technical%20Documenation%20v1.01.pdf

Contracts in Scope

./contracts/L1/L1AviERC721Bridge.sol
./contracts/L1/L1Bridge.sol
./contracts/L1/LiquidityPool.sol
./contracts/L2/L2AviERC721Bridge.sol
./contracts/L2/L2Bridge.sol
./contracts/libraries/AviPredeploys.sol

Contracts in Scope

./contracts/universal/AviBridge.sol

./contracts/universal/AviERC721Bridge.sol