// Private Smart Contract Security Assessment 06.17.2024 - 06.19.2024

L2 Governance Token

EtherFi

HALBORN

L2 Governance Token - EtherFi

Prepared by: HALBORN

Last Updated 06/24/2024

Date of Engagement by: June 17th, 2024 - June 19th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	0	1	2

TABLE OF CONTENTS

1. Introduction

`EtherFi` engaged Halborn to conduct a security assessment on their smart contracts beginning on 17th June and ending on 19th June. The security assessment was scoped to the smart contracts provided in the GitHub repository, commit hashes and further details can be found in the Scope section of this report.

This token contract is an ERC20 token with voting capabilities, token burning features, and owner-managed access control. It supports seamless upgrades to new implementations, ensuring long-term adaptability and security.

2. Assessment Summary

The team at Halborn was provided two days for the engagement and assigned one full-time security engineer to check 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 assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some minor security issues and recommendations that were mostly addressed by the `EtherFi team`.

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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 assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Static Analysis of security for scoped contract, and imported functions. (`Slither`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: ethfi-wormhole

(b) Assessed Commit ID: 9559e42

(c) Items in scope:

- src/EthfiL2Token.sol

Out-of-Scope: External libraries and financial-related attacks., Changes that occur outside of the scope of PRs.

REMEDIATION COMMIT ID:

- 037d292037d292

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
TWO STEPS OWNERSHIP	LOW	SOLVED - 06/20/2024
ATTACK VECTOR ON THE APPROVE/TRANSFERFROM METHODS	INFORMATIONAL	SOLVED - 06/20/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NEW OPCODES ARE NOT SUPPORTED BY ALL CHAINS (SOLIDITY VERSION >= 0.8.20)	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 TWO STEPS OWNERSHIP

// LOW

Description

The ownership of the contracts can be lost as the `EthfiL2Token` contract inherits from the `OwnableUpgradeable` contract, and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

```
    /**
     * @dev Transfers ownership of the contract to a new account
(`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner
{
        if (newOwner == address(0)) {
            revert OwnableInvalidOwner(address(0));
        }
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account
(`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        OwnableStorage storage $ = _getOwnableStorage();
        address oldOwner = $._owner;
        $._owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:P/S:U (2.5)

Recommendation

Consider using the `Ownable2StepUpgradeable` library over the `OwnableUpgradeable` library or implementing similar two-step ownership transfer logic into the contract.

Remediation Plan

SOLVED: The **Ether.Fi team** solved this issue by inheriting the `Ownable2StepUpgradeable` library instead of the `OwnableUpgradeable` library.

Remediation Hash

https://github.com/etherfi-protocol/ethfi-wormhole/commit/037d292a17f1edd9a72c3c9bf48a0e67f7140a55

7.2 ATTACK VECTOR ON THE APPROVE/TRANSFERFROM METHODS

// INFORMATIONAL

Description

The ERC20 standard is susceptible to an approval race condition, where simultaneous calls to `approve` or `transferFrom` can result in incorrect allowance values. This vulnerability arises when a spender's allowance is modified before the current transaction is completed, potentially allowing unintended double spending.

BVSS

AO:A/AC:H/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (1.7)

Recommendation

To mitigate this issue, it's recommended to use the `increaseAllowance` and `decreaseAllowance` functions instead of `approve`, or adopt the newer `permit` function from ERC20's EIP-2612 extension, which combines approval and transfer into a single atomic operation.

Remediation Plan

SOLVED: The **Ether.Fi team** solved this issue by implementing customized `increaseAllowance` and `decreaseAllowance` functions.

Remediation Hash

https://github.com/etherfi-protocol/ethfi-wormhole/commit/037d292a17f1edd9a72c3c9bf48a0e67f7140a55

7.3 NEW OPCODES ARE NOT SUPPORTED BY ALL CHAINS (SOLIDITY VERSION >= 0.8.20)

// INFORMATIONAL

Description

Solc compiler `version 0.8.20` switches the default target EVM version to Shanghai. The generated bytecode will include **PUSH0** opcodes. The recently released Solc compiler `version 0.8.25` switches the default target EVM version to Cancun, so it is also important to note that it also adds-up new opcodes such as **TSTORE**, **TLOAD** and **MCOPY**.
Be sure to select the appropriate EVM version in case you intend to deploy on a chain apart from mainnet, like L2 chains that may not support **PUSH0**, **TSTORE**, **TLOAD** and/or **MCOPY**, otherwise deployment of your contracts will fail.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Check compatibility of `EtherFi` target L2 chains and proceed to fix an appropriate compiler-pragma and `OpenZeppelin` version.

Remediation Plan

ACKNOWLEDGED: The **Ether.Fi team** is willing to only deploy this contract in Arbitrum and checked that ArbOS 20 Altas supports the majority of the Cancun EIPs, including EIP-1153 (TSTORE and TLOAD) and EIP-5656 (MCOPY), while PUSH0 was already supported before this upgrade.

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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, Slither was run on the all-scoped contracts. 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.

Slither Results

No major issues found by Slither.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.