Aragon - TokenRedemption

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **October 9th, 2023 - October 11th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	10/09/2023
0.2	Document Updates	10/11/2023
0.3	Draft Version	10/12/2023
0.4	Draft Review	10/13/2023
1.0	Remediation Plan	10/31/2023
1.1	Remediation Plan Review	11/02/2023
1.2	Remediation Plan	11/02/2023

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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Aragon engaged Halborn to conduct a security assessment on their smart contracts beginning on October 9th, 2023 and ending on October 11th, 2023. The security assessment was scoped to the smart contracts provided in the [redacted] GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 2 days for the engagement and assigned a team of 1 full-time security engineer[s] to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, which were mostly addressed by Aragon. The main one was the following:

- Fix pragma version.

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 assessment. 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 the security best practices. The following phases and associated tools were used during the assessment:

- 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.
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

2. 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.

2.1 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:

Exploitability 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
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:

$$E = \prod m_e$$

2.2 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)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

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

2.3 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.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	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

2.4 SCOPE

Code repositories:

1. Token Redemption

- Smart contracts in scope:
    1. TokenRedemption.sol (0x80fBB6122b8E023988e640dB1ae348a10A7933E8)
    2. Blocklist.sol (0xd650D03A93f23e4E92C4a8E4fef5d6953E5be01d)

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) FEE-ON-TRANSFER OR DEFLATIONARY TOKENS ARE NOT TAKEN INTO ACCOUNT	Informational (1.7)	ACKNOWLEDGED
(HAL-02) FLOATING PRAGMA	Informational (0.0)	SOLVED - 10/24/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) FEE-ON-TRANSFER OR DEFLATIONARY TOKENS ARE NOT TAKEN INTO ACCOUNT - INFORMATIONAL (1.7)

Description:

When fee-on-transfer or deflationary tokens are used for the redemption, the redeem function assumes that the deposited amount of tokens is the same as sent in the parameter. This could lead to a wrong swap calculation if a deflationary token is used, as the calculated ether amount could be higher than the real amount.

Code Location:

Listing 1: src/TokenRedemption.sol (Line 91)

```
80   function redeem(bytes32 _termsHash, uint256 tokenAmount) public {
81     if (blocklist.isBlocklisted(msg.sender)) {
82       revert AddressBlocklisted();
83     }
84     if (termsHash != _termsHash) {
85       revert TermsNotCorrect();
86     }
87     if (block.timestamp > endingTimestamp) {
88       revert RedemptionPeriodFinished();
89     }
90
91     token.transferFrom(msg.sender, address(this), tokenAmount);
92     uint256 ethToSend = (tokenAmount * tokenToEthRate) / tokenAmountBase;
93     (bool result, ) = msg.sender.call{ value: ethToSend }("");
94     if (!result) {
95       revert FailedToSendEth();
96     }
97
98     emit EthClaimed(msg.sender, tokenAmount, ethToSend);
99   }
```

BVSS:

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

Recommendation:

When calculating the ethToSend amount use the difference in balance before and after the token transfer.

Remediation Plan:

The Aragon team has accepted the risk, as the contracts will not be used with fee-on-transfer or deflationary tokens.

4.2 (HAL-02) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts 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, an outdated compiler version that might introduce bugs that affect the contract system negatively.

BVSS:

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

Recommendation:

Set the pragma to a fixed version.

Remediation Plan:

The Aragon team has solved the issue by fixing the pragma version in the commit ID: beb76441f9eb786787d25971302f16ef1b43a169.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

5.1 PROPER SWAP CALCULATION

Description:

The redeem function was tested to ensure a proper calculation in the swap ratio.

Results:

The swap was properly calculated except for fee-on-transfer or deflationary tokens.

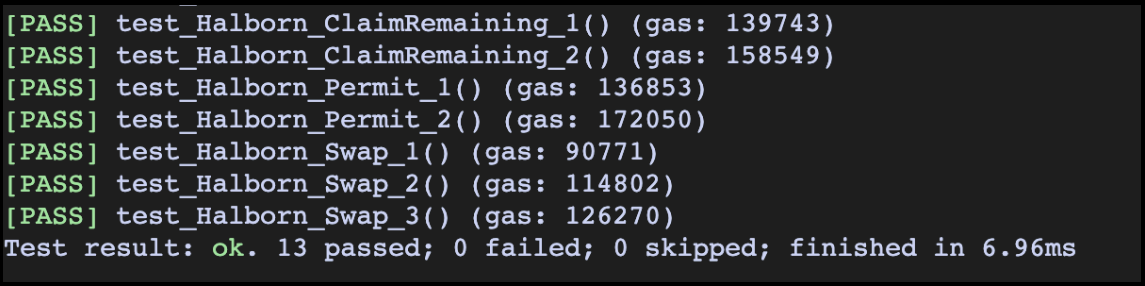
5.2 REMAINING AMOUNTS

Description:

The claimRemainings function was checked to ensure funds were properly distributed only after the end timestamp.

Results:

Remaining could not be distributed before the end timestamp, tokens were sent to the dead address while ether was sent back to the receiver address.



```
[PASS] test_Halborn_ClaimRemaining_1() (gas: 139743)
[PASS] test_Halborn_ClaimRemaining_2() (gas: 158549)
[PASS] test_Halborn_Permit_1() (gas: 136853)
[PASS] test_Halborn_Permit_2() (gas: 172050)
[PASS] test_Halborn_Swap_1() (gas: 90771)
[PASS] test_Halborn_Swap_2() (gas: 114802)
[PASS] test_Halborn_Swap_3() (gas: 126270)
Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 6.96ms
```

MANUAL TESTING

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the 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.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

Slither results for TokenRedemption	
Finding	**Impact**
TokenRedemption.blocklist (src/TokenRedemption.sol#20) is never initialized. It is used in: - TokenRedemption.redeem(bytes32,uint256) (src/TokenRedemption.sol#80-99)	High
TokenRedemption.termsHash (src/TokenRedemption.sol#19) is never initialized. It is used in: - TokenRedemption.redeem(bytes32,uint256) (src/TokenRedemption.sol#80-99)	High
Contract locking ether found: Contract TokenRedemption (src/TokenRedemption.sol#12-117) has payable functions: - TokenRedemption.receive() (src/TokenRedemption.sol#114-116) But does not have a function to withdraw the ether	Medium
TokenRedemption.constructor(IERC20Permit,uint256,address,uint256,bytes32,Blocklist)._recipient (src/TokenRedemption.sol#35) lacks a zero-check on : - recipient = _recipient (src/TokenRedemption.sol#42)	Low

Finding	Impact
TokenRedemption.claimRemainings() (src/TokenRedemption.sol#104-112) uses timestamp for comparisons Dangerous comparisons: - block.timestamp <= endingTimestamp (src/TokenRedemption.sol#105)	Low
TokenRedemption.redeem(bytes32,uint256) (src/TokenRedemption.sol#80-99) uses timestamp for comparisons Dangerous comparisons: - block.timestamp > endingTimestamp (src/TokenRedemption.sol#87)	Low
End of table for TokenRedemption	

Results summary:

The findings obtained as a result of the Slither scan were reviewed. The majority of Slither findings were determined false-positives.

THANK YOU FOR CHOOSING

// HALBORN