SECURITY AUDIT REPORT

SOMA Token

DATE

14 January 2025

PREPARED BY

0xTeam.

WEB3 AUDITS

info@0xTeam.Space

0xTeamSpace

0xTeamSpace

Contents

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	09 Jan 2025	K.Bob M.Gowda	Initial Audit Report
2.0	14 Jan 2025	K.Bob M.Gowda	Final Audit Report

0xTeam conducted a comprehensive Security Audit on the SOMA smart contracts to ensure the overall code quality, security, and correctness. The review focused on ensuring that the code functions as intended, identifying potential vulnerabilities, and safeguarding the integrity of SOMA's operations against possible attacks.

Report Structure

The report is divided into two primary sections:

1. **Executive Summary** : Provides a high-level overview of the audit findings.
2. **Technical Analysis** : Offers a detailed examination of the smart contract code.

Note :
The analysis is static and manual, exclusively focused on the smart contract code. The information provided in this report should be used to assess the security, quality, and expected behavior of the code.

1.0 Disclaimer

This is a summary of our audit findings based on our analysis, following industry best practices as of the date of this report.However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. The audit focuses on smart contract coding practices and any issues found in the code, as detailed in this report. For a complete understanding of our analysis, you should read the full report. We have made every effort to conduct a thorough analysis, but it's important to note that you should not rely solely on this report and cannot make claims against us based on its contents. We strongly advise you to perform your own independent checks before making any decisions. Please read the disclaimer below for more information.

DISCLAIMER: By reading this report, you agree to the terms outlined in this disclaimer. If you do not agree, please stop reading immediately and delete any copies you have. This report is for informational purposes only and does not constitute investment advice. You should not rely on the report or its content, and 0xTeam and its affiliates (including all associated companies, employees, and representatives) are not responsible for any reliance on this report.The report is provided "as is" without any guarantees. 0xTeam excludes all warranties, conditions, or terms, including those implied by law, regarding quality, fitness for a purpose, and use of reasonable care. Except where prohibited by law, 0xTeam is not liable for any type of loss or damage, including direct, indirect, special, or consequential damages, arising from the use or inability to use this report. The findings are solely based on the smart contract code provided to us.

2.0 Executive Summary

2.1 Overview

0xTeam has meticulously audited the SOMA smart contract project from 10 Oct 2024 to 18 Oct 2024. The primary objective of this audit was to assess the security, functionality, and reliability of the smart contracts before their deployment on the blockchain. The audit focused on identifying potential vulnerabilities, evaluating the contract's adherence to best practices, and providing recommendations to mitigate any identified risks. The comprehensive analysis conducted during this period ensures that the SOMA is robust and secure, offering a reliable environment for its users.

2.2 Scope

The scope of this audit involved a thorough analysis of the SOMA Smart Contract, focusing on evaluating its quality, rigorously assessing its security, and carefully verifying the correctness of the code to ensure it functions as intended without any vulnerabilities.

Files in Examination:

Contract Address	NA
Contracts In-Scope	• Contracts/TransparentUpgradeableProxy.sol

OUT-OF-SCOPE: External Solidity smart contract, other imported smart contracts.

2.3 Audit Summary

Name	Verified	Audited	Vulnerabilities
SOMA	Yes	Yes	Refer Section 5.0

2.4 Vulnerability Summary

High	Medium	Low	Informational
1	2	1	0

High Medium Low Informational

2.5 Summary of Findings

ID	Title	Severity	Fixed
[H-01]	Admin Account Misuse Risk	HIGH	✔
[M-01]	Lack of Input Validation in upgradeToAndCall	MEDIUM	✔
[M-02]	Potential Gas Optimization in _fallback Logic	MEDIUM	✔
[L-01]	Missing Event Emission for Admin Changes	LOW	✔

2.6 Recommendation Summary

Severity

Issues	High	Medium	Low	Informational
Open	1	2	1	0
Resolved	1	2	1	
Acknowledged				
Partially Resolved				

- **Open:** Unresolved security vulnerabilities requiring resolution.
- **Resolved:** Previously identified vulnerabilities that have been fixed.
- **Acknowledged:** Identified vulnerabilities noted but not yet resolved.
- **Partially Resolved:** Risks mitigated but not fully resolved.

3.0 Checked Vulnerabilities

We examined the smart contract for widely recognized and specific vulnerabilities. Below are some of the common vulnerabilities considered.

Category	Check Items
Source Code Review	➔ Reentrancy Vulnerabilities ➔ Ownership Control ➔ Time-Based Dependencies ➔ Gas Usage in Loops ➔ Transaction Sequence Dependencies ➔ Style Guide Compliance ➔ EIP Standard Compliance ➔ External Call Verification ➔ Mathematical Checks ➔ Type Safety ➔ Visibility Settings ➔ Deployment Accuracy ➔ Repository Consistency
Functional Testing	➔ Business Logic Validation ➔ Feature Verification ➔ Access Control and Authorization ➔ Escrow Security ➔ Token Supply Management ➔ Asset Protection ➔ User Balance Integrity ➔ Data Reliability ➔ Emergency Shutdown Mechanism

4.0 Techniques , Methods & Tools Used

The following techniques, methods, and tools were used to review all the smart contracts

- **Structural Analysis:**
  EThis involves examining the overall design and architecture of the smart contract. We ensure that the contract is logically organised, scalable, and follows industry best practices. This step is crucial for identifying potential structural issues that could lead to vulnerabilities or maintenance challenges in the future.

- **Static Analysis:**
  Static analysis is conducted using automated tools to scan the contract's codebase for common vulnerabilities and security risks without executing the code. This process helps identify issues such as reentrancy, arithmetic errors, and potential denial-of-service (DOS) vulnerabilities early on, allowing for quick remediation.

- **Code Review / Manual Analysis:**
  A manual, in-depth review of the smart contract's code is performed to verify the logic and ensure it matches the intended functionality as described in the project's documentation. During this phase, we also confirm the findings from the static analysis and check for any additional issues that may not have been detected by automated tools.

- **Dynamic Analysis:**
  Dynamic analysis involves executing the smart contract in various controlled environments to observe its behaviour under different conditions. This step includes running comprehensive test cases, performing unit tests, and monitoring gas consumption to ensure the contract operates efficiently and securely in real-world scenarios.

- **Tools and Platforms Used for Audit:**
  Utilising tools such as Remix , Slither, Aderyn, Solhint for static analysis, and platforms like Hardhat and Foundry for dynamic testing and simulation.

Note: The following values for "Severity" mean:

- High: Direct and severe impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal impact on the funds or the protocol's main functionality.
- Informational: Suggestions related to good coding practices and gas efficiency.

5.0 Technical Analysis

High Severity Issues

Issue #1 [Resolved]
[H-1] Admin Account Misuse Risk

Severity
HIGH

Location	Functions
Contracts/TransparentUpgradeableProxy.sol	➔ `_beforeFallback()`

Issue Description
The *_beforeFallback* function ensures that the admin cannot access the fallback function by reverting if *msg.sender == _getAdmin()*. However, this does not prevent the admin from accidentally or maliciously calling functions in the implementation contract directly. If the admin account is used for purposes other than managing the proxy (e.g., interacting with the proxied contract), it could lead to unintended behavior or vulnerabilities.

Issue Impact
The admin account has elevated privileges and is critical to the security of the proxy. If the admin account is compromised or misused, it could result in unauthorized upgrades, changes to the implementation, or even loss of funds.

Recommended Mitigation Steps & Code
To mitigate this risk, the admin account should be a dedicated account (e.g., an instance of ProxyAdmin) that is never used for any other purpose. Additionally, the contract should include clear documentation warning against using the admin account for anything other than administrative actions.

```
// Ensure the admin account is dedicated and not reused for other
purposes.
// Add a comment in the constructor to emphasize this:

constructor(
```

```
    address _logic,
    address admin_,
    bytes memory _data
) payable ERC1967Proxy(_logic, _data) {
      require(admin_ != address(0), "Admin address cannot be zero");
     _changeAdmin(admin_);
}
```

Medium Severity Issues

Issue #2 [Resolved]

[M-1] Lack of Input Validation in upgradeToAndCall

Severity
MEDIUM

Location	Functions
Contracts/TransparentUpgradeableProxy.sol	➔ upgradeToAndCall

Issue Description
The upgradeToAndCall function allows the admin to upgrade the implementation and execute arbitrary code in the new implementation. However, there is no validation to ensure that the newImplementation address is a valid contract or adheres to the expected interface (e.g., IERC1822Proxiable). This could lead to upgrading to a malicious or incompatible implementation.

Issue Impact
Upgrading to an invalid or malicious implementation could brick the proxy, cause unexpected behavior, or allow attackers to exploit vulnerabilities in the new implementation.

Recommended Mitigation Steps & Code
Add input validation to ensure that the newImplementation address is a valid contract and implements the required interface. For example:

```
function upgradeToAndCall(address newImplementation, bytes calldata data) external payable ifAdmin {
     require(Address.isContract(newImplementation), "New implementation must be a contract");
     require(IERC1822Proxiable(newImplementation).proxiableUUID() == addressA, "Invalid implementation");
     _upgradeToAndCall(newImplementation, data, true);
}
```

Issue #3 [Resolved]

[M-2] Potential Gas Optimization in _fallback Logic

Severity
MEDIUM

Location	Functions
Contracts/TransparentUpgradeableProxy.sol	➔ `_fallback`

Issue Description
The _fallback function delegates calls to the implementation contract. However, there is no optimization to minimize gas usage during delegation. For example, unnecessary checks or redundant logic could increase gas costs.

Issue Impact
Higher gas costs could make interactions with the proxy more expensive, especially for users executing frequent transactions.

Recommended Mitigation Steps & Code
Optimize the _fallback function to reduce gas consumption. For example, ensure that only necessary checks are performed before delegating the call:

```
function _fallback() internal virtual override {
     if (msg.sender != _getAdmin()) {
         super._fallback();
     } else {
         revert("TransparentUpgradeableProxy: admin cannot fallback to proxy
target");
     }
}
```

Low Severity Issues

Issue#4 [Resolved]
[L-1] Missing Event Emission for Admin Changes

Severity
LOW

Location	Functions
Contracts/TransparentUpgradeableProxy.sol	➔ `changeAdmin`

Issue Description
The changeAdmin function modifies the admin address but does not emit an event to log this change. While the _changeAdmin function emits an AdminChanged event, it is not explicitly documented or emphasized in the changeAdmin function.

Issue Impact
Without an event, it becomes harder to track admin changes on-chain, which could complicate auditing and monitoring.

Recommended Mitigation Steps & Code
Add an explicit comment in the changeAdmin function to clarify that an event is emitted. Alternatively, you can add a custom event for additional clarity:

```
function upgradeToAndCall(address newImplementation, bytes calldata data) external payable ifAdmin {
     require(Address.isContract(newImplementation), "New implementation must be a contract");
    require(IERC1822Proxiable(newImplementation).proxiableUUID() == aaddressA, "Invalid implementation");
    _upgradeToAndCall(newImplementation, data, true);
}
```

6.0 Auditing Approach and Methodologies Applied

The Solidity smart contract was audited using a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of the code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analysing the overall architecture of the Solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, signatures, and deprecated functions.
- **Documentation and comments**: Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behaviour and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices**: We checked that the code follows best practices and coding standards that are recommended by the Solidity community and industry experts. This ensures that the Solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum (Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract's security and performance.

Throughout the audit of the smart contracts, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behaviour. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimised for performance.

6.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the Solidity smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analysed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

6.2 Tools Used for Audit

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Remix,Slither, Aderyn, Solhint for Static Analysis and Hardhat & Foundry for Dynamic Analysis. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. 0xTeam takes pride in utilising these tools, which significantly contribute to the quality, security, and maintainability of our codebase

7.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the SOMA Project and methods for exploiting them. 0xTeam recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while 0xTeam considers the major security vulnerabilities of the analysed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the SOMA Solidity smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change. 0xTeam makes no undertaking to supplement or update this report based on changed circumstances or facts of which 0xTeam becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that 0xTeam use certain software or hardware products manufactured or maintained by other vendors. 0xTeam bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, 0xTeam does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by 0xTeam for the exclusive benefit of SOMA and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between 0xTeam and SOMA governs the disclosure of this report to all other parties including product vendors and suppliers.