CLIENT

CLIENT Contract Review

Version: 1.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the CLIENT smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the CLIENT smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the CLIENT smart contracts.

Overview

To be filled

Security Assessment Summary

This review was conducted on the files hosted on the CLIENT repository and were assessed at commit SOME_COMMIT.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 3 issues during this assessment. Categorised by their severity:

- Low: 1 issue.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the CLIENT smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
PREF-01	Unintended token burning due to invalidation of `_to` in transfer functions	Low	Resolved
PREF-02	ERC20 standard compliance	Informational	Closed
PREF-03	Miscellaneous General Comments	Informational	Open

PREF-01	Unintended token burning due to invalidation of _to in transfer functions		
Asset	FantomToken.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

The `_to` field in transfer events is not checked for the `0x0` address. External third-party applications which implement the ERC20 interface may interpret no-input in ERC20 fields as `0`. Thus users can quite easily inadvertently send tokens to the `0x0` address by forgetting to add a `_to` address in their third-party application. This is evident by the large number of tokens currently associated with the `0x0` address.

Recommendations

It is increasingly common for ERC20 tokens to include measures that ensure transfers to the `0x0` address are not possible. Validation of the `_to` field is recommended. This can be implemented in the `transfer()` and `transferFrom()` functions in the `ERC20Token` contract or in the analogous functions in the `Fantomtoken` contract (as the latter call the former).

Resolution

The recommendation has been implemented in commit c5339bd.

PREF-02	ERC20 standard compliance
Asset	`FantomToken.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

This section details the compliance with the ERC20 Standard [3] and adds any additional ERC20-related notes. Non-compliance with the ERC20 standard does not pose any security risk, however may cause issues with third-party applications which expect the standard.

The `FantomToken` contract complies with the ERC20 token standard with the following discrepancies:

- The `decimals` variable in `FantomToken` is a `uint256` rather than the specified `uint8`.

It should also be noted, that the ERC20 implementation has a known vulnerability to front-running in the `approve()` function [4].

Recommendations

Modify the `decimals` type to comply with the standard.

Be aware of the front-running issues in `approve()`, potentially add extended approve functions which are not vulnerable to the front-running vulnerability for future third-party-applications. See the Open-Zeppelin [5] solution for an example.

Resolution

The development team are aware of the issue but have decided not to implement a fix with the following comments.

We are aware of this issue however approval is only intended to be used by smart contracts which will not implement the double spending attack.

PREF-03	Miscellaneous General Comments
Asset	`contracts/*`
Status	Open
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **One sentance title.**
   Details of issue if more lines are needed.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `brownie` framework was used to perform these tests and the output is given below.

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].

[3] ERC-20 Token Standard. Github, Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[4] ERC20 API: An Attack Vector on Approve/TransferFrom Methods. Google Docs, 2018, Available: https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#heading=h.m9fhqynw2xvt.

[5] OpenZeppelin StandardToken.sol. Github, 2018, Available: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/v2.1.2/contracts/token/ERC20/ERC20.sol.