NonceAudits

Security Assessment

Aurus

TokenizedGold.sol

Smart Contracts Audit

Date: 9th October, 2022

Table of Content

1. Introduction

This Audit Report mainly focuses on the overall security of **TokenizedGold.sol**. With this report, we have tried to ensure the reliability and correctness of their smart contract by a complete and rigorous assessment of their system's architecture and the smart contract codebase.

1.1 Auditing Approach and Methodologies

The NonceAudit team has performed rigorous analysis of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is well structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find any potential issues like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks. In Automated Testing, we tested the Smart Contract with industry standard tools to identify vulnerabilities and security flaws.

The audit approach included:

- Analyzing the complexity of the code in-depth and detailed, manual review of the code, line-by-line.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

1.2 Audit Details

Project Name : Aurus
ID: ARS
Git commit hash: a7c15579c0a7cb3917d0aa02e5447596c91ce47e
Languages: Solidity (Smart contracts)
Platforms and Tools: Remix IDE, Solhint, VScode, Slither, Mythril

2. Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient, and working according to the specifications. The audit activities can be grouped into the following three categories:

2.1. Security

Identifying security-related issues within each contract and the system of contract.

2.2. Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

2.3. Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity

3. Security

Every issue in this report was assigned a severity level from the following:

High-level severity issues

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

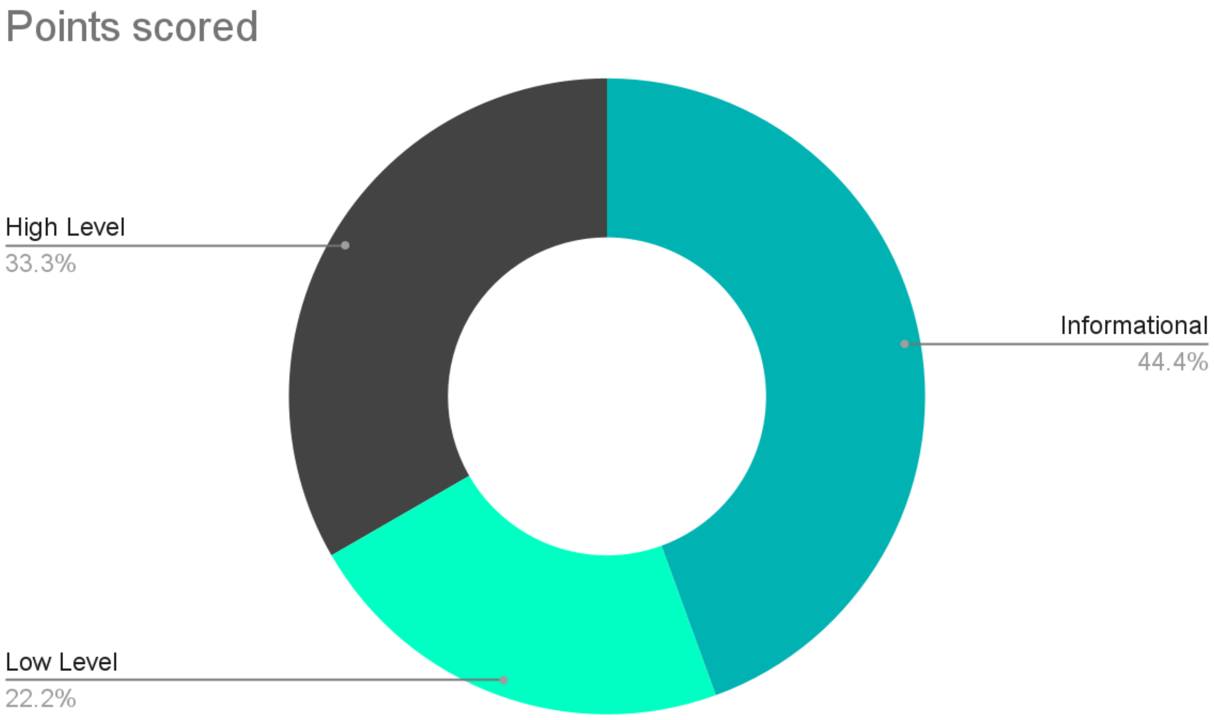
Low-level severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed.

Informational-level severity issues

Issues on this level are informational details that can remain unfixed but would be better fixed.

4. Vulnerability Summary



ID	Title	Severity	Status
ARS-01	Undeclared Identifier	High Level	CLOSED
ARS-02	State Variable Shadowing	High Level	CLOSED
ARS-03	Arithmetic overflow	High Level	CLOSED
ARS-04	Missing events arithmetic	Low Level	CLOSED
ARS-05	Missing Zero Address Validation	Low Level	CLOSED
ARS-06	Incorrect versions of Solidity	Informational Level	CLOSED
ARS-07	Conformance to Solidity naming conventions	Informational Level	CLOSED
ARS-08	Public functions that could be declared external	Informational Level	CLOSED
ARS-09	Modifiers and events	Informational Level	CLOSED

5. Manual Audit

For this section, the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM to test the contract functionality.

High-level severity issues

- **Title:** Undeclared Identifier
- **ID: ARS-01**
- **Line of Code:** L119
- **status:** CLOSED
- **Description:** An identifier called `_balances(address)` is used in `_transfer(address, address, uint)` but was never declared.
- **Recommendation:** Declare that identifier by creating a mapping to keep track of balances.
- **NOTE:** this error prevents compiling and deploying the contract so we added the following mapping to proceed auditing `mapping(address => uint256) public _balances`

- **Title:** Arithmetic overflow
- **ID: ARS-03**
- **Line of Code:** L107
- **status:** CLOSED
- **Description:** After solidity 0.8, overflows and underflows use the REVERT opcode instead of the INVALID opcode. Failing assertions and other internal checks like division by zero or arithmetic overflow do not use the invalid opcode but instead the revert opcode.
- **Recommendation:** Unfortunately there is no way to achieve what you want without wrapping your code in an `unchecked` block. Otherwise we suggest that you use a version of solidity less than 0.8 and keep using `SafeMathUpgradable`.

Medium level severity issues

- Not Found

Low Level Severity issues

- Not Found

- **Informational-level severity issues**
    - **Title:** Modifiers and events
    - **ID: ARS-08**
    - **Line of Code:** L144 / L151
    - **status:** **CLOSED**
    - **Description:** `onlyMinter()` and `ForceTransfer(address indexed, address indexed, uint256, bytes32)` are both declared at the end of contract
    - **Recommendation:** it is good practice to declare modifiers and events at the beginning of the contract for better readability.

6. Automated Audit

6.1 Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our Remix IDE for this analysis. Several violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contract.

```
12:9   warning  Error message for require is too long
27:81  warning  Visibility modifier must be first in list of modifiers
41:5   warning  Constant name must be in capitalized SNAKE_CASE
49:9   warning  Error message for require is too long
79:9   warning  Error message for require is too long
129:9  warning  Error message for require is too long
130:9  warning  Error message for require is too long
134:9  warning  Error message for require is too long
```

6.2 Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

High-level severity issues

- **Title:** State Variable Shadowing
- **ID: ARS-02**
- **Line of Code:** L18
- **status:** CLOSED
- **Description:** `_balances` is shadowing an inherited variable from `ERC20Upgradeable` contract which holds the same name.
- **Recommendation:** Remove the state variable shadowing

Medium level severity issues

- Not Found

Low Level Severity issues

- **Title:** Missing events arithmetic
- **ID: ARS-04**
- **Line of Code:** L34
- **status:** CLOSED
- **Description:** Detected missing events for critical arithmetic parameters. In `setMinterAddress(address)`
- **Recommendation:** Emit an event for critical parameter changes.

- **Title:** Missing Zero Address Validation
- **ID: ARS-05**
- **Line of Code:** L34 / L60
- **status:** CLOSED
- **Description:** Bob can call `setMinterAddress(address)` or `setFeeCollectionWallet(address)` without specifying the right addresses(ex. Null address), so Bob might lose fees or give access to an unauthorized minter.
- **Recommendation:** Check that the address is not zero.

- **Informational-level severity issues**
  - **Title:** Incorrect versions of Solidity
  - **ID: ARS-06**
  - **Line of Code:** L3
  - **status: CLOSED**
  - **Description:** `solc` frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex `pragma` statements.
  - **Recommendation:** Deploy with any of the following Solidity versions:
    - 0.7.5 - 0.7.6
    - 0.8.4 - 0.8.7 Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

  - **Title:** Conformance to Solidity naming conventions
  - **ID: ARS-07**
  - **Line of Code:** L20
  - **status: CLOSED**
  - **Description:** Solidity defines a naming convention that should be followed.
  - **Recommendation:** Follow the Solidity naming convention.

  - **Title:** Public functions that could be declared external
  - **ID: ARS-08**
  - **Line of Code:** L12 / L33 / L49 / L59 / L63 / L93 / L100
  - **status: CLOSED**
  - **Description:** `public` functions that are never called by the contract should be declared `external` to save gas.
  - **Recommendation:** Use the `external` attribute for functions never called from the contract.

6.3 Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities.

- **High-level severity issues**
  - Not Found

- **Medium level severity issues**
  - Not Found

- **Low Level Severity issues**
  - Not Found

- **Informational-level severity issues**
  - Not Found

7. Disclaimer

NonceAudit audit is not a security warranty, investment advice, or an endorsement of Aurus contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

8. Summary

The use case of the smart contract is simple and the code is relatively normal. Altogether, the code is written and demonstrates effective use of ERC20PausableUpgradeable and OwnableUpgradeable.

NonceAudits

www.nonceaudits.com