Smart Contract Security Audit Report

Deenar Token

1. Contents

2. General Information

This report contains information about the results of the security audit of the Deenar Token (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 22/08/2024 to 23/08/2024.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: deen-contracts. Initial review was done for the commit 235644.

The following contracts have been tested:

- contracts/GoldToken.sol
- contracts/TimeLock.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered several medium level issues. We also provided recommendations for resolving low-risk issues and adhering to best practices.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 26, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Burn function is not present	contracts/GoldToken.sol	Medium	Not fixed
Scheduled tx can't be cancelled	contracts/TimeLock.sol	Medium	Not fixed
The `removeOracle` function doesn't notify if the oracle is not found	contracts/GoldToken.sol	Low	Not fixed
Lack of the oracle sanity checks	contracts/GoldToken.sol	Low	Not fixed
Event not emitted	contracts/GoldToken.sol	Low	Not fixed
Variable can be set as immutable	contracts/TimeLock.sol	Info	Not fixed
Redundant function	contracts/GoldToken.sol	Info	Not fixed
Unnecessary `value` variable passed	contracts/TimeLock.sol	Info	Not fixed
`setOracle` optimization	contracts/GoldToken.sol	Info	Not fixed
Unused errors	contracts/GoldToken.sol	Info	Not fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Burn function is not present

Risk Level: Medium

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The GoldToken is an ERC20 token that is backed by physical gold. New tokens are minted whenever additional gold is added to the reserve. However, to maintain the balance between the token supply and the physical gold reserve, tokens should be burned if the amount of gold in the reserve decreases. Currently, there is no function to burn tokens, which could lead to an imbalance between the token supply and the underlying gold reserves.

Remediation:

To ensure that the token supply accurately reflects the physical gold reserves, consider adding a burn function. This function would allow tokens to be destroyed, reducing the total supply when gold is removed from the reserve.

5.2. Scheduled tx can’t be cancelled

Risk Level: Medium

Status: Not fixed

Contracts:

- contracts/TimeLock.sol

Description:

The issue in the TimeLock contract is that while it is possible to create and execute a pending transaction, there is no mechanism to cancel an ongoing pending transaction. This lack of a cancellation

mechanism could be problematic in various scenarios where it becomes necessary to halt or reverse a pending transaction before execution.

Remediation:

To address this issue, consider adding a separate role, such as `CANCELLER_ROLE`, similar to the approach used in the OpenZeppelin TimelockController. This role would allow designated users to cancel pending transactions, providing an additional layer of control and safety.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/governance/TimelockController.sol

5.3. The `removeOracle` function doesn't notify if the oracle is not found

Risk Level: Low

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The `removeOracle()` function does not currently handle the case where the specified oracle is not found in the list or set of oracles. This can lead to situations where the function is called, but the intended oracle is not removed, and the caller may be unaware that the operation was unsuccessful.

Remediation:

Consider reverting `false` or reverting the transaction if the oracle wasn't removed during the call. This will provide feedback to the caller, ensuring they are aware of the success or failure of the operation.

5.4. Lack of the oracle sanity checks

Risk Level: Low

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The `initialize()` function in the GoldToken contract allows for the setup of oracle addresses. However, there is no check to ensure that the input oracle addresses are non-zero or that the same oracle is not added multiple times.

```
contract GoldToken is
    Initializable,
    ERC20Upgradeable,
    AccessControlUpgradeable,
    UUPSUpgradeable
{
    ...
    function initialize(
        address defaultAdmin, // Owner grant roles
        address minter, // Admin
        address upgrader, // timeLock contract
        address[] memory _oracles
    ) public initializer {
    ...
     for (uint i = 0; i < _oracles.length; i++) {
            // @audit-issue same oracle can be pushed twice or can be equal to
zero
            oracles.push(_oracles[i]);
            _grantRole(ORACLE_ROLE, _oracles[i]);
        }
    ...
    }
    ...
}
```

Remediation:

Consider adding a check that input oracle address is non zero and not already in the array.

5.5. Event not emitted

Risk Level: Low

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The code includes a declared event `MintLimitUpdated`, but this event is not emitted anywhere in the code:

```
event MintLimitUpdated(uint256 newLimit);
```

Remediation:

To make use of this event, consider emitting it during the `setMintLimitByOracle()` function call. Additionally. it would be beneficial to modify the event to also include the oracle address that initiated the change, allowing for better tracking and auditing.

5.6. Variable can be set as immutable

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/TimeLock.sol

Description:

The `EXECUTION_DELAY` variable cannot be changed but is not set as `immutable`.

Remediation:

Since `EXECUTION_DELAY` is not expected to change during the execution of the contract, it is a good practice to define it as an immutable variable. This can be achieved by adding the `immutable` keyword before the variable declaration. By doing so, the variable will be set at compile-time instead of run-time, reducing gas costs and improving performance.

5.7. Redundant function

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The `oracles` array is declared as public, which means it can be accessed directly by external contracts and users. Therefore, creating a separate function to read the array is redundant:

```
    function getOracles() external view returns (address[] memory) {
        return oracles;
    }
```

Remediation:

Since the `oracles` array is already public, the `getOracles()` function is unnecessary and can be removed. Users can access the `oracles` array directly without needing an additional function.

5.8. Unnecessary `value` variable passed

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/TimeLock.sol

Description:

In the TimeLock contract, the `executeTransaction()` function includes a `value` parameter, which allows for the transfer of ETH as part of the transaction execution. However, since `executeTransaction()` is not marked as payable and there is no mechanism for sending ETH to the TimeLock contract, the `value` parameter is unnecessary.

Remediation:

To simplify the function, remove the `value` parameter and the associated logic for sending ETH.

5.9. `setOracle` optimization

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The `setOracle()` function could be optimized for better performance and renamed to `addOracle()` for clarity. Additionally, using OpenZeppelin's EnumerableSet can help manage oracles more efficiently, reducing gas costs and improving readability.

Remediation:

Here's how you can refactor the `setOracle()` function for better performance and clarity:

Without EnumerableSet:

```
function addOracle(address newOracle) external onlyRole(TIME_LOCKER_ROLE) {
    require(newOracle != address(0), ErrorZeroAddress());
    if (oracles.length != 0) {
        for (uint i = 0; i < oracles.length; i++) {
            if (oracles[i] == newOracle) {
                revert("Oracle already added");
            }
        }
    }
    oracles.push(newOracle);
    _grantRole(ORACLE_ROLE, newOracle);
    emit OracleAdded(newOracle);
}
```

Using EnumerableSet:

To further optimize, use OpenZeppelin's EnumerableSet:

```
import "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";

contract GoldToken is
    Initializable,
    ERC20Upgradeable,
    AccessControlUpgradeable,
    UUPSUpgradeable
{
    using EnumerableSet for EnumerableSet.AddressSet;

    EnumerableSet.AddressSet private oracles;

    event OracleAdded(address indexed newOracle);

    function addOracle(address newOracle) external onlyRole(TIME_LOCKER_ROLE)
{
        require(newOracle != address(0), "Oracle address cannot be zero");
```

```
        require(oracles.add(newOracle), "Oracle already added");

        _grantRole(ORACLE_ROLE, newOracle);
        emit OracleAdded(newOracle);
    }
}
```

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/structs/EnumerableSet.sol

5.10. Unused errors

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/GoldToken.sol

Description:

The codebase contains several unused custom errors:

```
error ErrorNotEnoughOracles();
error ErrorOracleAlreadyAdded();
error ErrorOracleNotFound();
```

Remediation:

To improve code clarity and maintainability, consider either:

1. **Using the Errors**: If these errors are intended to be part of the code's error-handling logic, integrate them into the appropriate places in the code where these conditions might occur.

   Example:

```
if (oracles[i] == newOracle) {
    revert ErrorOracleAlreadyAdded();
}
```

2. **Deleting the Errors**: If these errors are not necessary, remove them from the codebase.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.