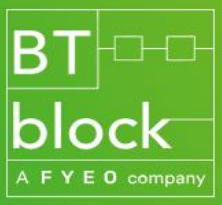

Security Assessment of The Sommelier Cellar Smart Contracts

PeggyJV

May 2022
Version 2.0

Presented by:
BTblock LLC

Corporate Headquarters
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Table of Contents

List of Figures

List of Tables

Executive Summary

Overview

PeggyJV engaged BTblock LLC to perform a Security Assessment of the Sommelier Cellar Smart Contracts.

The assessment was conducted remotely by the BTblock Security Team. Testing took place on April 04 - April 15, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the BTblock Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- BT-SOM-01 – All collected fees could be lost
- BT-SOM-02 – Platform charges too large fees
- BT-SOM-03 – The flag useUniswap is ignored

During the test, the following positive observations were noted regarding the scope of the engagement:

- The Sommelier team was very active and engaged in the review process. They were quick to answer any questions and were a pleasure to work with.

Based on formal verification we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

BTblock performed a Security Assessment of the Sommelier Cellar Smart Contracts. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/PeggyJV/cellar-contracts with the commit hash fb0578654a257ad89e2b5f30cbffe0bc5bfa02a6. A re-review of the recommended fixes took place on April 26, 2022 with the commit hash d885ab6157d3ad8aa18ced02c23027ab0a4b4029.

Files included in the code review

```
cellar-contracts/
├── contracts/
│   ├── interfaces/
│   │   ├── IAToken.sol
│   │   ├── IAaveIncentivesController.sol
│   │   ├── IAaveProtocolDataProvider.sol
│   │   ├── IAaveV2StablecoinCellar.sol
│   │   ├── IGravity.sol
│   │   ├── ILendingPool.sol
│   │   ├── IStakedTokenV2.sol
│   │   └── ISushiSwapRouter.sol
│   ├── utils/
│   │   └── MathUtils.sol
│   └── AaveV2StablecoinCellar.sol
├── src/
│   └── types/
│       └── common.ts
├── tasks/
│   ├── deploy/
│   │   └── aaveV2Cellar.ts
│   └── accounts.ts
├── tests/
│   └── AaveV2StablecoinCellar.test.ts
├── LICENSE
├── README.md
├── foundry.toml
├── hardhat.config.ts
├── package.json
├── remappings.txt
├── tsconfig.json
└── yarn.lock
```

Table 1: Scope

Technical Analyses and Findings

During the Security Assessment of the Sommelier Cellar Smart Contracts, we discovered:

- 2 findings with HIGH severity rating.
- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

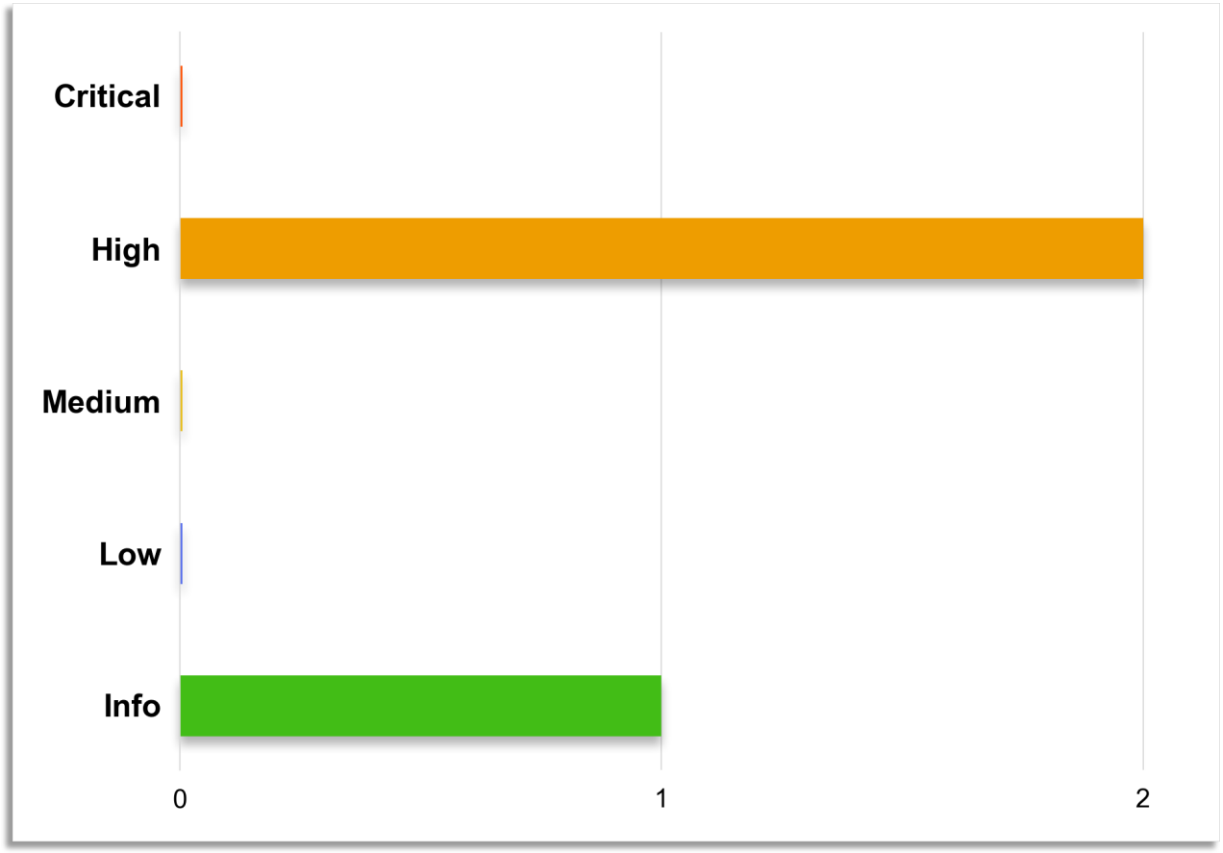

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
BT-SOM-01	High	All collected fees could be lost
BT-SOM-02	High	Platform charges too large fees
BT-SOM-03	Informational	The flag useUniswap is ignored

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality. Based on formal verification we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

`Aave Stablecoin Cellar` is a stablecoin strategy that maintains a lending position with the best performing stablecoin (highest interest rates, highest liquidity mining incentives, etc). The strategy will put the cellar liquidity into the DAI lending pool on Aave if the returns are best in DAI. If the market conditions change so that it is more profitable to put the cellar liquidity into the USDC lending pool on Aave, the strategy will move the cellar liquidity from DAI into USDC. Product logic is implemented on top of Ethereum network using `Solidity` smart contracts.

Summary of strengths

- The code and project files are well structured which makes them easy to read and maintain. The code is self-explanatory. The naming policy makes instructions understandable. All public functions are documented in comments.
- Well designed and clearly defined smart contracts access rights.
- The project has documentation that makes general product observation and describes interfaces and main logic.
- The contracts perform only the declared functionality.
- The main execution scenario has been covered with unit tests.
- Verification errors have custom explanation.
- The contracts are developed using the up-to-date compiler.

Summary of discovered issues and vulnerabilities

During the assessment 2 high and 1 informational severity issues were discovered. - High severity issues could lead to loss of collected fees and overcharging of fees.

Common recommendations

There are several project recommendations:

- Check the flag `isPaused` in the withdraw function. The withdraw function could be paused temporarily (e.g. for 3 days). It will help to mitigate the risk of exploiting any unknown vulnerability in the withdraw function. To avoid centralization, cool down logic could be implemented between calling the pause logic.
- Smart contract `AaveV2StablecoinCellar` could implement an Upgradable interface to have the ability to upgrade logic of the smart contracts.

All collected fees could be lost

Finding ID: BT-SOM-01
Severity: High
Status: Remediated

Description

It has been found that the `feesDistributor` variable is not initialized. This variable stores the address of the fee distributor on Cosmos network. In `Solidity`, this variable will be initialized with zero by default. This variable is used in the `transferFees()` function to transfer all accrued fees.

Proof of Issue

File name: contracts/AaveV2StablecoinCellar.sol
Line number: 773

```
gravityBridge.sendToCosmos(address(asset), feesDistributor, feeInAssets);
```

Severity and Impact Summary

In the case of calling the `transferFees()` function by the contract owner, all collected fees will be lost.

Recommendation

The `feesDistributor` variable should be initialized in the constructor of the `AaveV2StablecoinCellar` smart contract.

Platform charges too large fees

Finding ID: BT-SOM-02
Severity: High
Status: Remediated

Description

It has been found that the variable `lastTimeAccruedPlatformFees` is not updated after accrual of fees in the function `accrueFees()`. This variable is used to calculate platform fees that should be minted. To calculate the fees the following formula is used:

```
latform fees taken each accrual = activeAssets * (elapsedTime * (2% /
SECS_PER_YEAR))
```

where `elapsedTime` is calculated by the following formula:

```
uint256 elapsedTime = block.timestamp - lastTimeAccruedPlatformFees;
```

By design, the `accrueFees()` function will be called once per day. It could be called even every few seconds because the fee will be calculated only for elapsed time. The variable `lastTimeAccruedPlatformFees` is initialized only once in the smart contract constructor by the timestamp of the contract deployment. Actually, as `lastTimeAccruedPlatformFees` is not changed, elapsed time will be calculated for the whole period starting from the smart contract deployment time.

Proof of Issue

File name: contracts/AaveV2StablecoinCellar.sol
Line number: 671

```
uint256 elapsedTime = block.timestamp - lastTimeAccruedPlatformFees;
uint256 platformFeeInAssets =
    (_activeAssets() * elapsedTime * PLATFORM_FEE) / DENOMINATOR / 365 days;
```

Severity and Impact Summary

The contract will charge too large fees all the time. And more and more commissions every day.

Recommendation

It is recommended to update `lastTimeAccruedPlatformFees` variable in the `accrueFees()` function.

The flag useUniswap is ignored

Finding ID: BT-SOM-03
Severity: **Informational**

Description

It is found that a function `_swap` ignores flag `useUniswap` in the case of using single swap logic. In the case of a multihop swap, the flag `useUniswap` is used to choose between Sushiswap and Uniswap contracts. But in case of single swap logic, the Uniswap contract will be chosen even if the flag `useUniswap` returns false.

Proof of Issue

File name: contracts/AaveV2StablecoinCellar.sol
Line number: 1079

```
} else {
    // Single:

    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
        .ExactInputSingleParams({
            tokenIn: tokenIn,
            tokenOut: tokenOut,
            fee: SWAP_FEE,
            recipient: address(this),
            deadline: block.timestamp,
            amountIn: amountIn,
            amountOutMinimum: amountOutMinimum,
            sqrtPriceLimitX96: 0
        });

    // Executes a single swap on Uniswap.
    amountOut = uniswapRouter.exactInputSingle(params);
}
```

Severity and Impact Summary

The issue does not have a security impact, but is a logical mistake.

Recommendation

It is recommended to handle the flag `useUniswap`. It could be handled by failing the transaction or by the choosing between Sushiswap and Uniswap contracts.