F Y E O

Security Assessment of the Sommelier Cellar Smart Contracts

PeggyJV

July 2022
Version 2.2

Presented by:

FYEO Inc.

PO Box 147044
Lakewood CO 80214
United States

Table of Contents

List of Figures

List of Tables

Executive Summary

Overview

PeggyJV engaged FYEO Inc. to perform a Security Assessment of the Sommelier Cellar Smart Contracts.

The assessment was conducted remotely by the FYEO Security Team. Testing for the first security assessment took place on April 04 – April 15, 2022. Following further development by the Sommelier team, a second security assessment was performed on June 07 – June 20, 2022. Both reviews focused on the following objectives:

- To 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 results of our tests.

This report summarizes the engagement, tests performed, and findings that took place between June 07 – June 20, 2022. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified and should be prioritized for remediation to reduce to the risk they pose:

- FYEO-SOM-01 – Deposit ownership not checked
- FYEO-SOM-02 – Known issues in compiler
- FYEO-SOM-03 – Unused interfaces

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

Scope and Rules of Engagement

FYEO performed a Security Assessment of the Sommelier Cellar Smart Contracts between June 07 – June 20, 2022.

The source code was supplied through a public repository at https://github.com/PeggyJV/cellar-contracts with the commit hash 3915a3e067c899f6ac39311e44b8dbf975e4188c. A re-review of the recommended fixes took place on July 1, 2022.

- The remediation for the High severity finding FYEO-SOM-01 can be found with commit hash 72fbfa350d148f8db82b22062d6a20bc526f3c7a.
- The remediation for the Low severity finding FYEO-SOM-02 can be found with commit hash 6e84f0c0660e55974f204b0e6230f77f65565847.
- The remediation for the Informational finding FYEO-SOM-03 can be found with commit hash d9841c388a8f193b2eb364561c3791590d95e24e.

The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

Files included in the code review

```
cellar-contracts/
├── src/
│   ├── base/
│   │   ├── ERC4626.sol
│   │   └── Multicall.sol
│   ├── interfaces/
│   │   ├── IAToken.sol
│   │   ├── IAaveIncentivesController.sol
│   │   ├── IAaveProtocolDataProvider.sol
│   │   ├── IAaveV2StablecoinCellar.sol
│   │   ├── ICellarRouter.sol
│   │   ├── ICellarStaking.sol
│   │   ├── ICurveSwaps.sol
│   │   ├── IERC20.sol
│   │   ├── IERC4626.sol
│   │   ├── IGravity.sol
│   │   ├── ILendingPool.sol
│   │   ├── IMulticall.sol
│   │   ├── IStakedTokenV2.sol
│   │   └── ISushiSwapRouter.sol
│   ├── mocks/
│   │   ├── MockAToken.sol
│   │   ├── MockERC20.sol
│   │   ├── MockERC4626.sol
```

Files included in the code review

```
│   │   ├── MockGravity.sol
│   │   ├── MockIncentivesController.sol
│   │   ├── MockLendingPool.sol
│   │   ├── MockStkAAVE.sol
│   │   └── MockSwapRouter.sol
│   ├── utils/
│   │   └── Math.sol
│   ├── AaveV2StablecoinCellar.sol
│   ├── CellarRouter.sol
│   ├── CellarStaking.sol
│   └── Errors.sol
├── tasks/
│   ├── deploy/
│   │   ├── aaveV2Cellar.ts
│   │   ├── cellarRouter.ts
│   │   ├── index.ts
│   │   └── staking.ts
│   └── accounts.ts
├── foundry.toml
├── hardhat.config.ts
├── package-lock.json
├── package.json
└── remappings.txt
```

Table 1: Scope

Technical Analyses and Findings

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

- 1 finding with HIGH severity rating.
- 1 finding with LOW severity rating.
- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

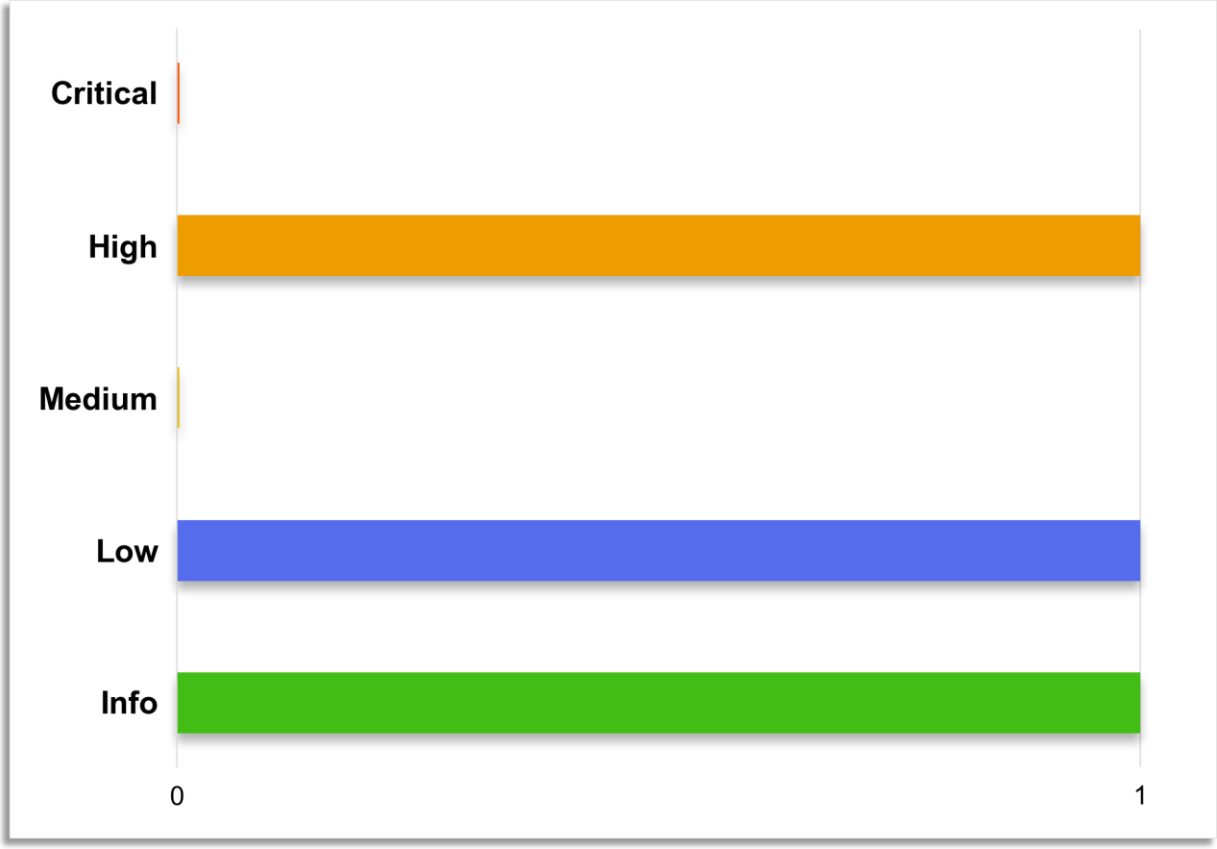

Figure 1: Findings by Severity

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
FYEO-SOM-01	High	Deposit ownership not checked
FYEO-SOM-02	Low	Known issues in compiler
FYEO-SOM-03	Informational	Unused interfaces

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 whose goal is to maintain a lending position with the best performing stablecoin (highest interest rates, highest liquidity mining incentives, etc). Smart contracts are implemented using Solidity for the Ethereum network. The product interacts with the following third-party smart contracts:

- AAVE Lending Pool
- AAVE Incentives Controller
- Sushi Swap Router
- Curve Pool Swap Registry
- Gravity Ethereum to Cosmos bridge

During the audit, it was found that AAVE has stopped offering incentives that make some reward logic outdated. We recommend considering a modification to the reward logic to support this.

The overall quality of the code is good: files are well structured, the naming policy makes instructions understandable, and the code is self-explanatory. The code uses `error` instead of `require` statements which make maintenance easier.

During the assessment, 1 high and 1 low severity and 1 informational issue were discovered. The high severity issue could lead to loss of deposit.

DEPOSIT OWNERSHIP NOT CHECKED

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

Description

The `depositAndSwapIntoCellar()` function accepts the `owner` address as a parameter without verification and applies the deposit to the `receiver` address which is a separate parameter. This logic can be exploited with a simple bot that:

- Listens to approve events on popular assets that may be deposited
- Catches the "approve" event and creates a `depositAndSwapIntoCellar` transaction where `receiver` is set to an attacker's address

Proof of Issue

File name: src/CellarRouter.sol

Line number: 88

```
function depositAndSwapIntoCellar(
        ERC4626 cellar,
        address[] calldata path,
        uint256 assets,
        uint256 assetsOutMin,
        address receiver,
        address owner
    ) public returns (uint256 shares) {
        ERC20 asset = cellar.asset();
        ERC20 assetIn = ERC20(path[0]);

        // Transfer assets from the user to the router.
        assetIn.safeTransferFrom(owner, address(this), assets);

//...
```

Severity and Impact Summary

The deposit may be stolen.

Recommendation

It is recommended to verify `owner` and `receiver` addresses. Example:

```
require(owner == msg.sender || owner == receiver, "INVALID_OWNER")
```

KNOWN ISSUES IN COMPILER

Finding ID: FYEO-SOM-02
Severity: Low
Status: Remediated

Description

There are several issues known to be present in solc 0.8.13, with some being introduced in this version.

Proof of Issue

File name: hardhat.config.ts

Line number: 108

```
const config: HardhatUserConfig = {
  solidity: {
    compilers: [
      {
        version: "0.8.13",
        settings: {
          optimizer: {
            enabled: optimizerEnabled,
            runs: 200,
            details: {
              // Enabled to fix stack errors when attempting to run test
coverage.
              yul: true,
              yulDetails: {
                stackAllocation: true,
              },
            },
          },
        },
      },
    ],
  },
```

Severity and Impact Summary

Solc 0.8.13 is affected by the following issues

- SOL-2022-2: NestedCallataArrayAbiReencodingSizeValidation (very low)
- SOL-2022-3: DataLocationChangeInInternalOverride (very low)
- SOL-2022-4: InlineAssemblyMemorySideEffects (medium)
- SOL-2022-5: DirtyBytesArrayToStorage (low)

Recommendation

It is recommended to use the version of compiler not affected by the medium issue (SOL-2022-4) by either upgrading it to “0.8.15” or downgrading it to a safe version. Please note that “0.8.15” is the most recent version at the moment and it may not be fully supported by different tools yet.

References

- https://docs.soliditylang.org/en/latest/bugs.html

UNUSED INTERFACES

Finding ID: FYEO-SOM-03
Severity: **Informational**
Status: **Remediated**

Description

Some interfaces are present in the repository but are not used in the code. `IAaveProtocolDataProvider.sol` seems to be a file from an older version. `IAToken.sol` is possibly meant to be used as a type for `assetAToken` variable (src/AaveV2StablecoinCellar.sol line 34).

Proof of Issue

File name: src/interfaces/IAaveProtocolDataProvider.sol

File name: src/interfaces/IAToken.sol

Severity and Impact Summary

There is no impact on security.

Recommendation

It is recommended to keep the code clean and remove unused interfaces.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

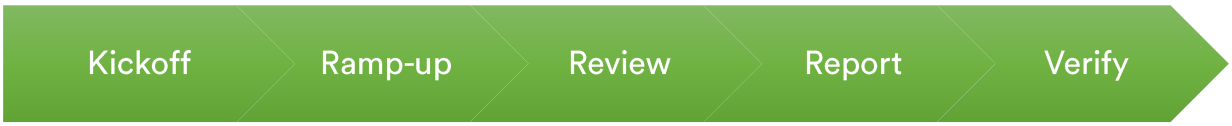

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking, and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations