# Veridise. Auditing Report

## Hardening Blockchain Security with Formal Methods

FOR


Veridise Inc.

June 27, 2022

► **Prepared For:**

Helio
https://helio.money

► **Prepared By:**

Bryan Tan
Jon Stephens
Jacob Van Geffen
Yanju Chen
Hongbo Wen

► **Contact Us:** contact@veridise.com

► **Version History:**

June 27, 2022 V1
June 17, 2022 Draft

# Contents



 

# 1 Executive Summary

From May 23 to June 13, Helio engaged Veridise to review the security of their Helio Protocol. The review covered the Helio DAO code, the Ceros code, and the reused parts of the MakerDAO smart contract code. Veridise conducted this assessment over 9 person-weeks, with 3 engineers working on code from commit `36ef1d2` to `97137ed` of the `helio-money/helio-smart-contracts` repository. The auditing strategy involved tool-assisted analysis of the source code performed by Veridise engineers. The tools that were used in the audit included a combination of static analyzers and bounded model checkers.

**Summary of issues detected.** The audit uncovered 42 issues, 7 of which are assessed to be of high or critical severity by Veridise auditors. The bugs discovered by Veridise can lead to a variety of undesired behaviors of the Helio protocol, including a denial of service attack that would prevent individuals from withdrawing their funds (V-HEL-VUL-001), a flashloan attack that could be used to gain all helio rewards (V-HEL-VUL-002), an initialization error that could allow the initial reward rate to be 100x the intended value (V-HEL-VUL-003) and missing auction functions that could allow funds to be locked in a stale auction (V-HEL-VUL-005). In addition to the high-severity bugs found, Veridise auditors also discovered a number of moderate severity issues, including a missing update to helio rewards before a user is liquidated (V-HEL-VUL-010).

**Code assessment.** The Helio Protocol is based on a fork of the Maker Protocol and shares much of the same infrastructure from that project. Specifically, the Helio Protocol is broken into three components: modified Multi-Collateral Dao (MCD), Helio DAO, and Ceros. The developers modified the MCD code* by removing several components. The Helio DAO code is a wrapper around the modified MCD and provides a convenient, but restricted, set of interfaces to the MCD code. The Ceros smart contracts implement (1) a sort of "auxiliary" collateral token, which we will refer to as "ceros tokens", that can be used as collateral in the Helio DAO; and (2) a wrapper that allows third-party tokens to be transparently used as collateral in the Helio DAO by automatically exchanging the third-party tokens for ceros tokens.

The client provided the source code of all three components for review. A test suite accompanied the source code; however, only a few of the tests in the test suite were passing while the audit was taking place. The client also provided a small slide deck on the motivation and goals of the Helio Protocol.

**Code Stability.** Over the period of the audit, new code was pushed to the repository 64 times, with the most recent commit occurring on June 16. Many of these commits involved bug fixes or updates to the test suite rather than new features. The Veridise auditors have therefore reviewed some portions of the code more than others.

* https://github.com/makerdao/dss

**Disclaimer.** We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

# 2 Project Dashboard

**Table 2.1:** Application Summary.

| Name | Version | Type | Platform |
|---|---|---|---|
| Helio Money | 36ef1d2 - 97137ed | Solidity | Ethereum |

**Table 2.2:** Engagement Summary.

| Dates | Method | Consultants Engaged | Level of Effort |
|---|---|---|---|
| May 22 - June , 2022 | Manual & Tools | 3 | 9 person-weeks |

**Table 2.3:** Vulnerability Summary.

| Name | Number | Resolved |
|---|---|---|
| Critical-Severity Issues | 1 | 1 |
| High-Severity Issues | 6 | 6 |
| Medium-Severity Issues | 4 | 4 |
| Low-Severity Issues | 14 | 14 |
| Warning-Severity Issues | 15 | 15 |
| Informational-Severity Issues | 2 | 2 |
| TOTAL | 42 | 42 |

**Table 2.4:** Category Breakdown.

| Name | Number |
|---|---|
| Logic Error | 15 |
| Maintainability | 7 |
| Access Control | 4 |
| Divide by Zero | 3 |
| Dead Code | 2 |
| Usability Issue | 2 |
| Arithmetic Overflow | 1 |
| Denial of Service | 1 |
| Flashloan | 1 |
| Gas Optimizations | 1 |
| Invalid Interface | 1 |
| Locked Funds | 1 |
| Transaction Order | 1 |
| Unused Return | 1 |
| Variable Shadowing | 1 |

# 3 Audit Goals and Scope

## 3.1 Audit Goals

The engagement was scoped to provide a security assessment of the Helio Protocol, particularly around their DAO contract. In our audit, we sought to answer the following questions:

- Can a user withdraw collateral without repaying their loan?
- Are users able to access their funds that are not used as collateral for a loan?
- Are helio rewards reflective of a user's stake in the protocol?
- Are users fairly rewarded for using the HAY stablecoin?
- Are users able to interact with the administrative contracts?
- Is the method of interacting with the contract aligned with the user's expectation?
- Is the deployment process straightforward and intuitive?
- Can user funds be locked or lost?

## 3.2 Audit Methodology & Scope

**Audit Methodology.** To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to evaluate how the code behaves given unexpected inputs. To do this, we created several unit tests using the Foundry testing framework, and then we applied the property-based testing capabilities of Foundry to fuzz potentially vulnerable methods.

*Scope*. To understand the scope of the audit, we first reviewed the Maker Protocol documentation (because the Helio Protocol is based on a fork of Maker) and focused our efforts on understanding the Maker Protocol components used by Helio. In this phase, our main goal was to understand how the Helio Protocol interacts with the Multi-Collateral Dao contracts. Afterwards, we assessed the other Helio Protocol contracts for bugs and security issues.

In terms of the scope of the audit, the key components we considered include the following:

- The Multi-Collateral Dao components used by Helio
- The Helio DAO deposit, borrow, payback, and withdraw methods
- The Ceros deposit and withdraw mechanisms
- The high level business logic

## 3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

**Table 3.1:** Severity Breakdown.

| | Somewhat Bad | Bad | Very Bad | Protocol Breaking |
|---|---|---|---|---|
| Not Likely | Info | Warning | Low | Moderate |
| Likely | Warning | Low | Moderate | High |
| Very Likely | Low | Moderate | High | Critical |

In this case, we judge the likelihood of a vulnerability as follows:

| | |
|---|---|
| Not Likely | A small set of users must make a specific mistake |
| Likely | Requires a complex series of steps by almost any user(s)<br>- OR -<br>Requires a small set of users to perform an action |
| Very Likely | Can be easily performed by almost anyone |

In addition, we judge the impact of a vulnerability as follows:

| | |
|---|---|
| Somewhat Bad | Inconveniences a small number of users and can be fixed by the user |
| Bad | Affects a large number of people and can be fixed by the user<br>- OR -<br>Affects a very small number of people and requires aid to fix |
| Very Bad | Affects a large number of people and requires aid to fix<br>- OR -<br>Disrupts the intended behavior of the protocol for a small group of users through no fault of their own |
| Protocol Breaking | Disrupts the intended behavior of the protocol for a large group of users through no fault of their own |

# Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

**Table 4.1:** Summary of Discovered Vulnerabilities.

| ID | Description | Severity | Status |
|---|---|---|---|
| V-HEL-VUL-001 | Denial of Service | Critical | Fixed |
| V-HEL-VUL-002 | Fraudulent Rewards | High | Fixed |
| V-HEL-VUL-003 | Initialization Error | High | Fixed |
| V-HEL-VUL-004 | Direct User Interaction | High | Fixed |
| V-HEL-VUL-005 | Insufficient Methods | High | Fixed |
| V-HEL-VUL-006 | User Bypass of Functions | High | Fixed |
| V-HEL-VUL-007 | No Token Check | High | Fixed |
| V-HEL-VUL-008 | Test Suite Problem | Medium | Acknowledged |
| V-HEL-VUL-009 | Liquidation Incentives | Medium | Intended Behavior |
| V-HEL-VUL-010 | No Rewards Update | Medium | Fixed |
| V-HEL-VUL-011 | Cannot Re-Enable Collateral | Medium | Intended Behavior |
| V-HEL-VUL-012 | Emit of Stop Event | Low | Fixed |
| V-HEL-VUL-013 | Update Failure | Low | Acknowledged |
| V-HEL-VUL-014 | Awkward Approval Criteria | Low | Intended Behavior |
| V-HEL-VUL-015 | Bugs Removing Users | Low | Fixed |
| V-HEL-VUL-016 | Ilk Initialization Failure | Low | Intended Behavior |
| V-HEL-VUL-017 | Affected Method Revert | Low | Fixed |
| V-HEL-VUL-018 | Awkward USB Approval | Low | Fixed |
| V-HEL-VUL-019 | Vat Permission Revoke | Low | Fixed |
| V-HEL-VUL-020 | Division by Zero #1 | Low | Fixed |
| V-HEL-VUL-021 | Division by Zero #2 | Low | Fixed |
| V-HEL-VUL-022 | gemJoin Permission Revoke | Low | Fixed |
| V-HEL-VUL-023 | No Initialization Check | Low | Fixed |
| V-HEL-VUL-024 | Multiple Rewards Grant | Low | Fixed |
| V-HEL-VUL-025 | Ignored Return Value | Low | Fixed |
| V-HEL-VUL-026 | Definition Mismatch | Warning | Fixed |
| V-HEL-VUL-027 | Multiplication Overflow | Warning | Acknowledged |
| V-HEL-VUL-028 | User Debt Error | Warning | Fixed |
| V-HEL-VUL-029 | Address Desync | Warning | Acknowledged |
| V-HEL-VUL-030 | Division by Zero #3 | Warning | Fixed |
| V-HEL-VUL-031 | Unused Flop/Flap Code | Warning | Fixed |
| V-HEL-VUL-032 | Add Atomic Allowance Methods | Warning | Fixed |
| V-HEL-VUL-033 | Ineffective Use of mulDiv | Warning | Fixed |
| V-HEL-VUL-034 | Rounding Error for Leap Years | Warning | Fixed |
| V-HEL-VUL-035 | Inconsistent Comment | Warning | Fixed |
| V-HEL-VUL-036 | Possible Interface Error | Warning | Acknowledged |
| V-HEL-VUL-037 | OpenZeppelin Version Usage | Warning | Fixed |
| V-HEL-VUL-038 | Burning of Stablecoin | Warning | Intended Behavior |
| V-HEL-VUL-039 | Shadowing Function Parameter | Warning | Acknowledged |
| V-HEL-VUL-040 | Incorrect State Variable Cast | Warning | Acknowledged |
| V-HEL-VUL-041 | Dead Code | Informational | Fixed |
| V-HEL-VUL-042 | Inconsistency with Interface | Informational | Acknowledged |

## 4.1 Detailed Description of Bugs

In this section, we describe each uncovered vulnerability in more detail.

### 4.1.1 V-HEL-VUL-001: Interaction withdraw denial of service

| | | | |
|---|---|---|---|
| **Severity** | Critical | **Commit** | e46d015 |
| **Type** | Denial of Service | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | withdraw() | | |

**Description** Since `GemJoin.join` is public, it is possible to perform a denial of service attack on the `Interaction.withdraw` method, which will prevent users from withdrawing their funds through the Interaction contract. This is because a user can deposit free collateral using `GemJoin.join`, which can then be withdrawn using `Interaction.withdraw`. Since this collateral was not deposited via the Interaction contract, `deposits[token]` will be less than the value deposited using `Interaction.deposit`. Users will therefore not be able to withdraw since the reduction of `deposits[token]` by `dink` will underflow.

#### Theoretical Attack Scenario

1. Users normally deposit collateral through `Interaction.deposit`. This increases the value of the `deposits[token]` state variable.
2. An attacker directly invokes the collateral's `GemJoin.join` method to transfer collateral to the vat.
3. The attacker calls `Interaction.withdraw` to withdraw their recently deposited collateral. This decreases the `deposits[token]` state variable.
4. The attacker repeats steps 2-4 to repeatedly decrease `deposits[token]` until it reaches 0.
5. Users can no longer use `Interaction.withdraw` to retrieve their collateral because the `deposits[token] -= dink` line will overflow.

```
1  function join(address usr, uint wad) external {
2      require(live == 1, "GemJoin/not-live");
3      require(int(wad) >= 0, "GemJoin/overflow");
4      vat.slip(ilk, usr, int(wad));
5      require(gem.transferFrom(msg.sender, address(this), wad),
6          "GemJoin/failed-transfer");
7      emit Join(usr, wad);
8  }
9
```

**Snippet 4.1:** Location of the join function that can be called directly to deposit funds

```
    function withdraw(
        address participant,
        address token,
        uint256 dink
    ) external returns (uint256) {
        ...

        // Collateral is actually transferred back to user inside ‘exit‘ operation.
        // See GemJoin.exit()
        collateralType.gem.exit(msg.sender, dink);
        deposits[token] -= dink;

        emit Withdraw(participant, dink);
        return dink;
    }

```

**Snippet 4.2:** Location where the funds are withdrawn and deposits is reduced

**Recommendation** If the intention is for a user to only interact with the Interaction contract, add access controls to the GemJoin functions. If it is intended to allow users to interact with the Maker contracts, remove deposits since it might not reflect the actual amount of deposited funds.

### 4.1.2 V-HEL-VUL-002: Fraudulent rewards can be generated with flashloan

| Severity | High | Commit | 36e7d41 |
|---|---|---|---|
| Type | Flashloan | Status | Fixed |
| Files | contracts/Interaction.sol | | |
| Functions | borrow() | | |

**Description** For any given ilk, the amount of rewards granted by HelioRewards is a function of (1) the amount of locked collateral that the rewarded user has at the time of the drop; and (2) the time elapsed since the last drop for that user. However, in `Interaction.borrow`, the reward is dropped after the user debt has been updated. Thus, an attacker can theoretically harvest a large number of rewards with a flashloan.

#### Theoretical Attack Scenario

1. An attacker initially uses a small amount X of collateral to borrow HAY, beginning the reward tracking.
2. The attacker waits for a long period of time, say T.
3. The attacker takes out a flashloan, deposits a large amount Y of collateral, and uses it to borrow HAY.
4. Because the reward drop occurs after the debt amount has been updated, the amount of reward drop now depends on X+Y over the time T instead of X.
5. The attacker pays back the loaned HAY with `payback` , withdraws the collateral, and then pays back the flashloan.

```
function borrow(address token, uint256 hayAmount) external returns (uint256) {
    ...

    vat.frob(collateralType.ilk, msg.sender, msg.sender, msg.sender, 0, dart);
    vat.move(msg.sender, address(this), hayAmount * RAY);
    hayJoin.exit(msg.sender, hayAmount);
    dropRewards(token, msg.sender);

    emit Borrow(msg.sender, hayAmount);
    return uint256(dart);
}
```

**Snippet 4.3:** The location where the user's debt is updated before rewards are dropped

**Recommendation** Moving the reward drop somewhere before `vat.frob` will avoid the problem, since the rewards will be calculated with respect to (1) the debt amount before the borrow; and (2) the time elapsed since the last drop.

### 4.1.3 V-HEL-VUL-003: Missing jug initialization

| | | | |
|---|---|---|---|
| **Severity** | High | **Commit** | 73b1cd1 |
| **Type** | Logical Error | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | setCollateralType() | | |

**Description** In `setCollateralType`, an ilk is initialized with `vat.init`, but there is no corresponding `jug.init` to set the rate. This means that the `duty` field of the ilks storage variable in the jug will be set to 0 for every ilk. Consequently:

- Since `base` is initially set to 0 and the rate calculated by `jug.drip` depends on the computation `base + ilks[ilk].duty`, the updated rate will be set to 0 when the deployer forgets to call `jug.file("base", ...)`. A rate of 0 is considered to be an uninitialized ilk, which causes methods such as `vat.frob` to revert.
- Even if the deployer calls `jug.file("base", ...)`, the time period is calculated with `block.timestamp() - ilks[ilk].rho`, where `ilks[ilk].rho` will be 0 on the first call to `jug.drip`. For the first call, the time period is the time passed since the Unix epoch (January 1, 1970). This means that the `rate` may increase significantly after the first call. For example, initializing `base` with 1000000003022266000000000000 will result in a `rate` of 1e27 before any call to `drip` and 1.483e29 after the call to `drip`.
- Due to the high initial rate, the multiplications of ilk `rate` with quantities in the `vat.frob` method will result in larger values and be more prone to overflow. enditemize Furthermore, increases to the rate will have less effect, since the increases will be very small compared to the initial rate.

```
1    function setCollateralType(
2        address token,
3        address gemJoin,
4        bytes32 ilk,
5        address clip
6    ) external auth {
7        vat.init(ilk);
8        enableCollateralType(token, gemJoin, ilk, clip);
9    }
```

**Snippet 4.4:** Location of the `setCollateralType` function

**Recommendation** We believe the proper use of jug is as follows:

- `setCollateralType` should call `jug.init` and then call `jug.file(ilk, "duty", ...)` to add argument to the rate for the given provided collateral type.
- The current deploy and test scripts set base to be "ONE" + some percentage. The correct way is to set a global rate with `jug.init("base", ...)`
- If only a global rate is desired, use `jug.file(ilk, "duty", 0)`
- Adjust the rates so that they are more consistent with the developers' intended reward schedule

 

### 4.1.4 V-HEL-VUL-004: Users can interact with vat directly

| | | | |
|---|---|---|---|
| **Severity** | High | **Commit** | e46d015 |
| **Type** | Access Control | **Status** | Fixed |
| **Files** | contracts/vat.sol | | |
| **Functions** | frob/flux/etc | | |

**Description** Similar to V-HEL-VUL-006, by using unprotected MakerDAO functions, it is possible for a user to interact with the vat (using the `frob` method in particular) to bypass Interaction.sol. Thus the state variables tracked in the Interaction contract (e.g., `usersInDebt` and `deposits`) may not reflect the actual state of the protocol. In addition, Interaction's state variables are intended to aid liquidators in finding individuals that are eligible for liquidation. It is therefore possible that liquidators will miss individuals who are eligible for liquidation but interacted with the vat directly.

```
    function frob(
        bytes32 i,
        address u,
        address v,
        address w,
        int dink,
        int dart
    ) external {
        ...
    }
```

**Snippet 4.5:** Location of the `frob` declaration in the Vat contract

**Recommendation** Restrict access to the Vat contract so that only authorized users may call its functions.

### 4.1.5 V-HEL-VUL-005: AuctionProxy needs more management methods

| | | | |
|---|---|---|---|
| **Severity** | High | **Commit** | c2b0aba |
| **Type** | Logical Error | **Status** | Fixed |
| **Files** | contracts/AuctionProxy.sol | | |
| **Functions** | N/A | | |

**Description** Although AuctionProxy wraps the MakerDAO auction logic (`Clipper`), it only provides the methods `startAuction` and `buyFromAuction`. While these functions allow users to create and buy from auctions, they do not capture all necessary auction functionality provided by the Clipper contract. Specifically, they do not allow an auction to be reset, which is required if an auction is not completed in a certain amount of time. As a result, funds can be locked in an auction and would require an admin to rescue them by interacting with the Clipper contract. In addition, the AuctionProxy contract does not provide functionality to check the status of or cancel an auction. Finally, it does not call `Clipper.upchost`, which the Maker protocol assumes will be called periodically to update the `chost` contract variable.

**Recommendation** Add the following functionality to AuctionProxy:

- Resetting an auction
- Checking the status of an auction
- Canceling an auction
- Call `upchost`

**Developer Response** The developers indicated that they do not intend to add functionality to cancel an auction.

### 4.1.6 V-PAR-HEL-006: Users can bypass AuctionProxy by using unprotected MakerDAO functions

| | | | |
|---|---|---|---|
| **Severity** | High | **Commit** | c2b0aba |
| **Type** | Access Control | **Status** | Fixed |
| **Files** | contracts/dog.sol, conracts/clip.sol | | |
| **Functions** | bark, kick, redo, take | | |

**Description** A user can bypass the AuctionProxy contract by instead directly calling functions in the MakerDAO contract, such as `bark`. These functions are currently marked as external with no access controls, allowing anyone to call them. Thus, the developers cannot enforce the additional behaviors provided by AuctionProxy. If a user were to interact with the Maker contracts, there can be negative consequences such as inaccessible liquidation funds.

```
    function bark(
        bytes32 ilk,
        address urn,
        address kpr
    ) external auth returns (uint256 id) {
        ...
    }
```

**Snippet 4.6:** Location the `bark` function, which can be called by anyone

**Recommendation** Restrict access to the Dog and Clipper contracts so that only authorized users may call its functions.

### 4.1.7 V-HEL-VUL-007: No check for existing tokens in setCollateralType

| | | | |
|---|---|---|---|
| **Severity** | High | **Commit** | 36e7d41 |
| **Type** | Locked Funds | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | setCollateralType | | |

**Description** It is possible to call `setCollateralType` for an existing token, but with a different ilk, causing an existing entry of `collaterals` to be overridden. This can lead to dangerous behavior, since changing the ilk can cause borrows to be locked in the overwritten ilk. To illustrate this, let's say ilk1 was overwritten by ilk2 . If a user borrowed from ilk1 and then tried to pay back the protocol after the switch to ilk2, upon the call to `vat.frob()` in `Interaction.payback()`, the call can revert when the `dart` is added to `ilk.art` due to an underflow. In addition, since the user's urn is located using the ilk (in this case essentially `urns[ilk2][msg.sender]`), the borrow also couldn't be located if the ilk's bytes32 identifier were changed.

```
    function setCollateralType(
        address token,
        address gemJoin,
        bytes32 ilk,
        address clip
    ) external auth {
        vat.init(ilk);
        collaterals[token] = CollateralType(GemJoinLike(gemJoin), ilk, 1, clip);
        IERC20Upgradeable(token).safeApprove(gemJoin, type(uint256).max);
        vat.rely(gemJoin);
        emit CollateralEnabled(token, ilk);
    }
```

**Snippet 4.7:** Location of the `setCollateralType` function that can overwrite the ilk for a token

**Recommendation** Don't allow an admin to change the ilk if there are any outstanding debts (likely don't allow it at all). If the ilk needs to be adjusted, instead change the underlying Ilk struct itself.

### 4.1.8 V-HEL-VUL-008: Problems with test suite

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | 36e7d41 |
| **Type** | Maintainability | **Status** | Acknowledged |
| **Files** | tests/* | | |
| **Functions** | N/A | | |

**Description** There are several problems with the test suite that prevent the developers from testing their code effectively or thoroughly.

- Not all of the tests in the test suite pass. Furthermore, some of the tests do not work at all. This means that it is harder to catch regressions resulting from changes to the source code.
- There is a significant amount of duplication in the test suite setup code.
- The test suite setup code and deployment scripts have duplicated code, which means that they are more likely to diverge as development continues.
- The test suite only provides partial code coverage and mostly consists of integration tests.

**Recommendation**

- Refactor the shared setup code into reusable helper functions, and call these helper functions inside of the test suite and the deployment scripts.
- Add smaller tests for the isolated components like CeVault and HelioRewards

### 4.1.9 V-HEL-VUL-009: Not all liquidation incentives are passed onto the user

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | c2b0aba |
| **Type** | Logical Error | **Status** | Intended Behavior |
| **Files** | contracts/AuctionProxy.sol | | |
| **Functions** | startAuction, buyFromAuction | | |

**Description** Incentives are currently passed onto the user by converting an internal HAY balance into tokens. To do so, the developers divide `vat.usb(address(this)` by `RAY`. As a result of this integer division, any incentives that are less than `RAY` will not be passed along to the user. This means that (1) the leftover incentives will be given to other liquidators who did not earn them; and (2) small purchases (`< RAY`) can result in no collateral being passed onto the liquidator. It is therefore possible for small loans to go unliquidated.

```
    function startAuction(
        address user,
        address keeper,
        IERC20 usb,
        UsbGemLike usbJoin,
        VatLike vat,
        DogLike dog,
        HelioProviderLike helioProvider,
        CollateralType calldata collateral
    ) external onlyDao returns (uint256 id) {
        uint256 usbBal = usb.balanceOf(address(this));
        id = dog.bark(collateral.ilk, user, address(this));

        usbJoin.exit(address(this), vat.usb(address(this)) / RAY);
        usbBal = usb.balanceOf(address(this)) - usbBal;
        usb.transfer(keeper, usbBal);

        ...
     }
```

**Snippet 4.8:** Location where incentives are forwarded to the keeper in `startAuction`

**Recommendation** Pass all rewards to the user via the vat.

**Developer Response** The developers indicated that since amounts `< RAY` are very small (less than one token), they do not intend to implement a fix for this issue.

### 4.1.10 V-HEL-VUL-010: startAuction does not update rewards

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | c2b0aba |
| **Type** | Logical Error | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | startAuction | | |

**Description** A liquidated user's Helio rewards are not updated once a liquidation is initiated via `startAuction`. Consequently, if a user were to borrow HAY and then neglect to interact with the contract until a liquidation was initiated, they would not be able to claim rewards for the liquidated funds. Had the user called `helioRewards.drop` before liquidation, however, they would have been rewarded for borrowing these funds.

```
    function startAuction(
        address user,
        address keeper,
        IERC20 usb,
        UsbGemLike usbJoin,
        VatLike vat,
        DogLike dog,
        HelioProviderLike helioProvider,
        CollateralType calldata collateral
    ) external onlyDao returns (uint256 id) {
        uint256 usbBal = usb.balanceOf(address(this));
        id = dog.bark(collateral.ilk, user, address(this));

        usbJoin.exit(address(this), vat.usb(address(this)) / RAY);
        usbBal = usb.balanceOf(address(this)) - usbBal;
        usb.transfer(keeper, usbBal);

        // Burn any derivative token (hBNB incase of ceabnbc collateral)
        if (address(helioProvider) != address(0)) {
            helioProvider.daoBurn(user, collateral.clip.sales(id).lot);
        }
    }
```

**Snippet 4.9:** Location of the `startAuction` function which does not drop rewards

**Recommendation** Drop helio rewards for the liquidated user in `startAuction`.

### 4.1.11 V-HEL-VUL-011: Cannot Re-Enable Collateral

| Severity | Medium | Commit | f46da2d |
|---|---|---|---|
| Type | Logical Error | Status | Intended Behavior |
| Files | contracts/Interaction.sol | | |
| Functions | setCollateralType, removeCollateralType | | |

**Description** In some cases, the developer may intend to toggle whether a collateral is live or not. The developer may attempt to do this by calling `removeCollateralType` to disable a collateral type and then calling `setCollateralType` (with the same ilk) to re-enable the collateral type. However, attempting to call `setCollateralType` again will always revert because the token / ilk pair has already been initialized. Thus, a collateral type cannot be enabled ever again once it has been removed, so users cannot withdraw their collateral.

```
    function setCollateralType(
        address token,
        address gemJoin,
        bytes32 ilk,
        address clip
    ) external auth {
        require(collaterals[token].live == 0, "Interaction/token-already-init");
        vat.init(ilk);
        jug.init(ilk);
        jug.file(ilk, "duty", 0);
        collaterals[token] = CollateralType(GemJoinLike(gemJoin), ilk, 1, clip);
        IERC20Upgradeable(token).safeApprove(gemJoin, type(uint256).max);
        vat.rely(gemJoin);
        emit CollateralEnabled(token, ilk);
    }
```

**Snippet 4.10:** Location of the `setCollateralType` function which initializes the ilk

**Recommendation** Add a method to toggle whether a collateral is live or not. This method should check that the token/ilk pair has already been initialized.

**Developer Response** The `removeCollateralType` function is intended to permanently remove an ilk and therefore should not be re-enabled via `setCollateralType`.

### 4.1.12 V-HEL-VUL-012: HelioRewards emits Stop event in start() method

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 36e7d41 |
| **Type** | Logical Error | **Status** | Fixed |
| **Files** | contracts/HelioRewards.sol | | |
| **Functions** | start | | |

The `start` function emits an event named `Stop`, even though the contract has a `Start` event.

```
function start() public auth {
    live = 1;
    emit Stop(msg.sender);
}
```

**Snippet 4.11:** Location of the `Stop` emit in the `start` function

**Recommendation** Emit the `Start` event rather than `Stop`.

### 4.1.13 V-HEL-VUL-013: buyFromAuction can impact Interaction state

| Severity | Low | Commit | e46d015 |
|---|---|---|---|
| Type | Logical Error | Status | Acknowledged |
| Files | Interaction.sol | | |
| Functions | buyFromAuction | | |

**Description** The Interaction contract tracks a token's cumulative deposits (`deposits`) and which users have borrowed funds (`usersInDebt`). While both of these values can be impacted upon a liqudation, neither value is updated in `buyFromAuction`. Thus, these values may not reflect the current state of the protocol, which can make it difficult for liquidators to find users eligible for liquidation.

```
    function buyFromAuction(
        address token,
        uint256 auctionId,
        uint256 collateralAmount,
        uint256 maxPrice,
        address receiverAddress
    ) external {
        CollateralType memory collateral = collaterals[token];
        IHelioProvider helioProvider = IHelioProvider(helioProviders[token]);
        auctionProxy.buyFromAuction(
            msg.sender,
            auctionId,
            collateralAmount,
            maxPrice,
            receiverAddress,
            usb,
            usbJoin,
            vat,
            helioProvider,
            collateral
        );
    }
```

**Snippet 4.12:** Location of the `buyFromAuction` function in the Interaction contract

**Recommendation** Update both of these values in `buyFromAuction`.

### 4.1.14 V-HEL-VUL-014: HelioProvider provideInABNBc() has awkward approval requirements

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | e46d015 |
| **Type** | Usability Issue | **Status** | Intended Behavior |
| **Files** | contracts/HelioProvider.sol, contracts/ceros/CerosRouter.sol | | |
| **Functions** | HelioProvider.provideInABNBc(),CerosRouter.depositABNBcFrom() | | |

**Description** The HelioProvider's `provideInABNBc` function calls `_ceRouter.depositABNBcFrom`, which invokes the `transferFrom` method on the aBNBc token contract. This means that the user must approve the `_ceRouter`, otherwise the transaction will revert. Consequently, users must be aware of HelioProvider's implementation details as it is unlikely that they will know to approve the CerosRouter rather than the HelioProvider.

```
    function provideInABNBc(uint256 amount)
    external
    override
    nonReentrant
    returns (uint256 value)
    {
        value = _ceRouter.depositABNBcFrom(msg.sender, amount);
        // deposit ceToken as collateral
        _provideCollateral(msg.sender, value);
        emit Deposit(msg.sender, value);
        return value;
    }
```

**Snippet 4.13:** Location of `provideInABNBc` which requires `_ceRouter` to perform the transfer

**Recommendation** Have the user approve the HelioProvider instead. The HelioProvider can transfer from the user to itself, and then the router can transfer tokens from the provider to the router. This should be safe because `depositABNBcFrom` is marked as `onlyProvider`.

**Developer Response** The developers indicated that fixing this issue would require changes to the behavior of the CerosRouter. Since this contract will be shared by several projects, the developers don't want to make adjustments to the interface.

### 4.1.15 V-HEL-VUL-015: Bugs when removing users from usersInDebt

| Severity | Low | Commit | e46d015 |
|---|---|---|---|
| Type | Logical Error | Status | Fixed |
| Files | contracts/Interaction.sol | | |
| Functions | payback | | |

**Description** The Interaction contract tracks the users who have borrowed funds to aid liquidators in finding users to liquidate. It is therefore the case that when a user pays back their loan, `usersInDebt` may require updating if the entire loan is paid off. While `usersInDebt` is updated in the payback function, there are several cases where it may not be updated accurately. First, the check for no debt is subject to rounding errors, which can prevent users from being removed from `usersInDebt` even after the loan is paid off. Second, a user is removed from `usersInDebt` when they have repaid all art in the corresponding urn. However, the case where a single user has multiple urns is not considered so a user may be removed from the set while they still have debt.

```
    function payback(address token, uint256 usbAmount) external returns (int256) {
        CollateralType memory collateralType = collaterals[token];
        // _checkIsLive(collateralType.live); Checking in the ‘drip‘ function

        IERC20Upgradeable(usb).safeTransferFrom(msg.sender,address(this),usbAmount);
        usbJoin.join(msg.sender, usbAmount);
        (,uint256 rate,,,) = vat.ilks(collateralType.ilk);
        (, uint256 art) = vat.urns(collateralType.ilk, msg.sender);
        int256 dart = int256(hMath.mulDiv(usbAmount, 10 ** 27, rate));
        if (uint256(dart) * rate < usbAmount * (10 ** 27) &&
            uint256(dart + 1) * rate <= vat.usb(msg.sender)
        ) {
            dart += 1;
            // ceiling
        }
        vat.frob(collateralType.ilk, msg.sender, msg.sender, msg.sender, 0, - dart);

        if ((int256(rate * art) / 10 ** 27) == dart) {
            EnumerableSet.remove(usersInDebt, msg.sender);
        }

        ...
    }
```

**Snippet 4.14:** Location where `usersInDebt` is updated in the Payback function

**Recommendation** First, check the urn when determining if a user has paid off their loan. Second, track user debt for each collateral type.

### 4.1.16 V-HEL-VUL-016: enableCollateralType does not initialize ilks

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 4997d0e |
| **Type** | Maintainability | **Status** | Intended Behavior |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | enableCollateralType | | |

**Description** Previously, the developers indicated that calling `enableCollateralType` on an existing token but with different ilks is a feature. However, if the ilk does not yet exist in the vat, it will not be initialized. This will cause future calls to `deposit` and `withdraw` to revert when they call `vat.frob()`. While this can be avoided by calling `setCollateralType` when using a new ilk, there are no checks to enforce the desired interaction.

```
    function enableCollateralType(
        address token,
        address gemJoin,
        bytes32 ilk,
        address clip
    ) public auth {
        collaterals[token] = CollateralType(GemJoinLike(gemJoin), ilk, 1, clip);
        IERC20Upgradeable(token).approve(gemJoin, type(uint256).max);
        vat.rely(gemJoin);
        emit CollateralEnabled(token, ilk);
    }
```

**Snippet 4.15:** Location of `enableCollateralType`

**Recommendation** Add a check to `enableCollateralType` to ensure that ilk exists in the vat.

**Developer Response** The intention is to call `setCollateralType` if the ilk has not been initialized by the vat. The developers indicated that they will ensure that the correct function is called.

### 4.1.17 V-HEL-VUL-017: AuctionProxy assumes permissions to USB/USB join

| Severity | Low | Commit | c2b0aba |
|---|---|---|---|
| Type | Maintainability | Status | Fixed |
| Files | contracts/AuctionProxy.sol | | |
| Functions | startAuction, buyFromAuction | | |

**Description** The AuctionProxy currently assumes that it has been authorized by the vat to move funds via the `usbJoin` contract. If the developers forget to grant this authorization, the `startAuction` and `buyFromAuction` functions will always revert.

```
1    function startAuction(...) external onlyDao returns (uint256 id) {
2        ...
3
4        usbJoin.exit(address(this), vat.usb(address(this)) / RAY);
5
6        ...
7    }
```

**Snippet 4.16:** Location where `startAuction` requires `usbJoin` authorization

```
1    function buyFromAuction(...) external onlyDao {
2        ...
3
4        usbJoin.exit(address(this), vat.usb(address(this)) / RAY);
5
6        ...
7    }
```

**Snippet 4.17:** Location where `buyFromAuction` requires `usbJoin` authorization

**Recommendation** Relying on developers to perform complex initialization can be error-prone. Grant the required permissions in the contract logic instead.

### 4.1.18 V-HEL-VUL-018: buyFromAuction involves awkward HAY approval process

| Severity | Low | Commit | c2b0aba |
|---|---|---|---|
| Type | Usability Issue | Status | Fixed |
| Files | contracts/Interaction.sol, contracts/AuctionProxy.sol | | |
| Functions | buyFromAuction | | |

**Description** The `buyFromAuction` function in the Interaction contract requires the caller to pay HAY in order to receive collateral. This payment occurs when the AuctionProxy invokes `transferFrom` in its `buyFromAuction` function. Since the payment occurs from the AuctionProxy contract though, the user must approve the AuctionProxy contract even though they invoke the transaction through the Interaction contract. Consequently, users must be aware of the Interaction contract's implementation details as it is unlikely they will know to approve AuctionProxy rather than Interaction.

```
    function buyFromAuction(
        address token,
        uint256 auctionId,
        uint256 collateralAmount,
        uint256 maxPrice,
        address receiverAddress
    ) external {
        CollateralType memory collateral = collaterals[token];
        IHelioProvider helioProvider = IHelioProvider(helioProviders[token]);
        auctionProxy.buyFromAuction(
            msg.sender,
            auctionId,
            collateralAmount,
            maxPrice,
            receiverAddress,
            usb,
            usbJoin,
            vat,
            helioProvider,
            collateral
        );
    }
```

**Snippet 4.18:** Location of the `buyFromAuction` function in the Interaction contract

**Recommendation** Make the AuctionProxy a library that is used by the Interaction contract.

### 4.1.19 V-HEL-VUL-019: buyFromAuction does not revoke vat permissions

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | c2b0aba |
| **Type** | Access Control | **Status** | Fixed |
| **Files** | contracts/AuctionProxy.sol | | |
| **Functions** | buyFromAuction | | |

**Description** The AuctionProxy contract's `buyFromAuction` function calls `vat.hope` on the collateral type's clip but never revokes the permissions by calling `vat.nope`. For security purposes, permissions should be revoked when they are no longer needed. Additionally, it is desirable to grant more limited permissions to reduce the impact of potential security vulnerabilities, so it is desirable to use `vat.behalf` rather than `vat.hope`.

```
    function buyFromAuction(
        address user,
        uint256 auctionId,
        uint256 collateralAmount,
        uint256 maxPrice,
        address receiverAddress,
        IERC20 usb,
        UsbGemLike usbJoin,
        VatLike vat,
        HelioProviderLike helioProvider,
        CollateralType calldata collateral
    ) external onlyDao {
        ...

        vat.hope(address(collateral.clip));

        ...
    }
```

**Snippet 4.19:** Location where vat permissions are granted via `vat.hope`

**Recommendation** Limit the permissions granted to AuctionProxy to only those that are required and revoke the permissions when they are no longer needed.

### 4.1.20 V-HEL-VUL-020: Divided by zero #1

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | c2b0aba |
| **Type** | Divide by Zero | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | estimatedLiquidationPrice | | |

**Description** The `estimatedLiquidationPrice` function can revert if the amount parameter is set to `-ink` since `uint256(int256(ink) + amount)` will cause `ink` to be 0.

```
    function estimatedLiquidationPrice(
        address token,
        address usr,
        int256 amount
    ) external view returns (uint256) {
        ...

        (uint256 ink, uint256 art) = vat.urns(collateralType.ilk, usr);
        require(amount >= -int256(ink), "Cannot withdraw more than current amount");
        if (amount < 0) {
            ink = uint256(int256(ink) + amount);
        } else {
            ink += uint256(amount);
        }

        ...

        return backedDebt / ink;
    }
}
```

**Snippet 4.20:** Location of the divide-by-zero issue in `estimatedLiquidationPrice`

**Recommendation** Check if `ink` is zero and handle this case appropriately.

### 4.1.21 V-HEL-VUL-021: Divided by zero #2

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | c2b0aba |
| **Type** | Divide by Zero | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | currentLiquidationPrice | | |

**Description** If the user has not deposited any funds, then the function will revert because `ink` will be 0.

```
1    function currentLiquidationPrice(
2        address token,
3        address usr
4    ) external view returns (uint256) {
5        CollateralType memory collateralType = collaterals[token];
6        _checkIsLive(collateralType.live);
7        (uint256 ink, uint256 art) = vat.urns(collateralType.ilk, usr);
8        (, uint256 rate,,,) = vat.ilks(collateralType.ilk);
9        (,uint256 mat) = spotter.ilks(collateralType.ilk);
10       uint256 backedDebt = (art * rate / 10 ** 36) * mat;
11       return backedDebt / ink;
12   }
```

**Snippet 4.21:** Location where `currentLiquidationPrice` reverts if `ink` is 0

**Recommendation** Add a check to determine if the `ink` is 0.

### 4.1.22 V-HEL-VUL-022: removeCollateralType does not revoke gemJoin permissions

| Severity | Low | Commit | c2b0aba |
|---|---|---|---|
| Type | Access Control | Status | Fixed |
| Files | contracts/Interaction.sol | | |
| Functions | removeCollateralType | | |

**Description** The `enableCollateralType` function calls `vat.rely` on the token's `gemJoin`. However, `removeCollateralType` does not call `vat.deny` to revoke the permissions. To reduce the impact of any potential vulnerabilities, permissions should be revoked when they are no longer needed.

```
1    function removeCollateralType(address token) external auth {
2        collaterals[token].live = 0;
3        address gemJoin = address(collaterals[token].gem);
4        IERC20Upgradeable(token).approve(gemJoin, 0);
5        emit CollateralDisabled(token, collaterals[token].ilk);
6    }
```

**Snippet 4.22:** Location of `removeCollateralType` which does not revoke `gemJoin` permissions

**Recommendation** Call `vat.deny` in `removeCollateralType`.

### 4.1.23 V-HEL-VUL-023: setRate does not check token initialization

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | c2b0aba |
| **Type** | Logical Error | **Status** | Fixed |
| **Files** | contracts/HelioRewards.sol | | |
| **Functions** | setRate | | |

**Description** The function `setRate` does not check that the argument token is initialized, so it is possible for the owner to set the rate on a non-existent token. If this were to occur, there will be no error messages, reverts, or warnings.

```
1    function setRate(address token, uint256 newRate) external auth {
2        Ilk storage pool = pools[token];
3        pool.rewardRate = newRate;
4    }
```

**Snippet 4.23:** Location of the `setRate` function

**Recommendation** Require that the token exists when setting the rate.

### 4.1.24 V-HEL-VUL-024: Reinitializing pool causes rewards to be granted multiple times

| Severity | Low | Commit | c2b0aba |
|---|---|---|---|
| Type | Logical Error | Status | Fixed |
| Files | contracts/HelioRewards.sol | | |
| Functions | initPool | | |

**Description** If the owner calls the `initPool` function multiple times for the same token, then the token will be duplicated in the `poolsList` array. Consequently, rewards will be awarded multiple times for the same token due to the duplicate entries.

```
1    function initPool(address token, bytes32 ilk, uint256 rate) external auth {
2        pools[token] = Ilk(rate, block.timestamp, ilk);
3        poolsList.push(token);
4    }
```

**Snippet 4.24:** Location of the `initPool` function

**Recommendations** Developers should add logic to handle calls to initPool on tokens that are already initialized.

### 4.1.25 V-HEL-VUL-025: Approve's return value ignored

| Severity | Low | Commit | c2b0aba |
|---|---|---|---|
| Type | Unused Return | Status | Fixed |
| Files | contracts/Interaction.sol | | |
| Functions | enableCollateralType, removeCollateralType | | |

**Description** An IERC20 contract's `approve` function returns a boolean that indicates if the approval was successful. While approve typically succeeds, different token implementations might have different conditions that need to be fulfilled to be approved. By not checking the return value, it is possible to list a collateral type where users cannot deposit their collateral.

```
1    function enableCollateralType(
2        address token,
3        address gemJoin,
4        bytes32 ilk,
5        address clip
6    ) public auth {
7        collaterals[token] = CollateralType(GemJoinLike(gemJoin), ilk, 1, clip);
8        IERC20Upgradeable(token).approve(gemJoin, type(uint256).max);
9        vat.rely(gemJoin);
10       emit CollateralEnabled(token, ilk);
11   }
```

**Snippet 4.25:** Location where `approve` is called in `enableCollateralType`

```
1    function removeCollateralType(address token) external auth {
2        collaterals[token].live = 0;
3        address gemJoin = address(collaterals[token].gem);
4        IERC20Upgradeable(token).approve(gemJoin, 0);
5        emit CollateralDisabled(token, collaterals[token].ilk);
6    }
```

**Snippet 4.26:** Location where `approve` is called in `removeCollateralType`

**Recommendation** Check the return value of `approve` calls.

### 4.1.26 V-HEL-VUL-026: VatLike interface in jug does not match definition of vat

| Severity | Warning | Commit | 73b1cd1 |
|---|---|---|---|
| Type | Invalid Interface | Status | Fixed |
| Files | contracts/jug.sol | | |
| Functions | VatLike | | |

**Description** The return type of the `ilks` function of the VatLike interface in jug.sol does not match the return type of `ilks` in vat.sol.

```
interface VatLike {
    function ilks(bytes32) external returns (
        uint256 Art,   // [wad]
        uint256 rate   // [ray]
    );
    function fold(bytes32,address,int) external;
}
```

**Snippet 4.27:** Location of the VatLike interface declaration in jug.sol

```
contract Vat is VatLike {
    struct Ilk {
        uint256 Art;   // Total Normalised Debt     [wad]
        uint256 rate;  // Accumulated Rates         [ray]
        uint256 spot;  // Price with Safety Margin  [ray]
        uint256 line;  // Debt Ceiling              [rad]
        uint256 dust;  // Urn Debt Floor            [rad]
    }

    mapping (bytes32 => Ilk) public ilks;

    ...
}
```

**Snippet 4.28:** Location of ilks function provided by the public `ilks` contract variable in vat.sol

**Recommendation** Rather than declaring a new VatLike interface in jug.sol, import the VatLike interface that Vat inherits from where the declaration is correct.

### 4.1.27 V-HEL-VUL-027: Opportunities to avoid multiplication overflow

| Severity | Warning | Commit | 36e7d41 |
|---|---|---|---|
| Type | Arithmetic Overflow | Status | Acknowledged |
| Files | contracts/AuctionProxy.sol, contracts/ceros/CeVault.sol | | |
| Functions | buyFromAuction, _deposit and _withdraw | | |

**Description** The developers' hMath library can be used to avoid overflows in computations with multiplications and divisions. Currently, there are a few locations where multiplications and divisions are directly used rather than the hMath library.

```
1    function buyFromAuction(...) external onlyDao {
2        ...
3
4        uint256 usbMaxAmount = (maxPrice * collateralAmount) / RAY;
5
6        ...
7    }
```

**Snippet 4.29:** Location in `AuctionProxy.buyFromAuction` that may overflow

```
1    function _deposit(...) private returns (uint256) {
2        uint256 ratio = _aBNBc.ratio();
3        _aBNBc.transferFrom(msg.sender, address(this), amount);
4        uint256 toMint = (amount * 1e18) / ratio;
5
6        ...
7    }
```

**Snippet 4.30:** Location in `CeVault._deposit` that may overflow

```
1    function _withdraw(...) private returns (uint256) {
2        uint256 ratio = _aBNBc.ratio();
3        uint256 realAmount = (amount * ratio) / 1e18;
4
5        ...
6    }
```

**Snippet 4.31:** Location in `CeVault._withdraw` that may overflow

**Recommendation** Use the hMath library for computations that involve multiplications and divisions to avoid overflows.

 

### 4.1.28 V-HEL-VUL-028: User added to usersInDebt in deposit

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | e46d015 |
| **Type** | Logical Error | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | deposit | | |

**Description** The Interaction contract tracks the users who have debt to allow liquidators to iterate over individuals who may require liquidation. Currently a user is added to the set `usersInDebt` upon a deposit, but at this point a user has no debt. This pollutes the set and can make it more difficult to discover users eligible for liquidation.

```
1    function deposit(
2        address participant,
3        address token,
4        uint256 dink
5    ) external returns (uint256) {
6        ...
7
8        EnumerableSet.add(usersInDebt, participant);
9
10       emit Deposit(participant, dink);
11       return dink;
12   }
```

**Snippet 4.32:** Location where a user is added to `usersInDebt` in the `deposit` function

**Recommendation** Add a user to `usersInDebt` in the `borrow` function.

### 4.1.29 V-HEL-VUL-029: Potential HelioProvider and CerosRouter address desync

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | 119cbd0 |
| **Type** | Logical Error | **Status** | Acknowledged |
| **Files** | contracts/ceros/CerosRouter.sol, contracts/ceros/HelioProvider.sol | | |
| **Functions** | N/A | | |

**Description** In CerosRouter contract, it is possible to change the provider with the `changeProvider` function; however, there is no analogous function in HelioProvider that allows the `_ceRouter` to be changed. This means that it is possible for the HelioProvider to reference a CerosRouter with a different provider.

```
1    function changeProvider(address provider) external onlyOwner {
2        _provider = provider;
3        emit ChangeProvider(provider);
4    }
```

**Snippet 4.33:** Location of the `changeProvider` function in CerosRouter

**Recommendation** Add a function to change a provider's `_ceRouter`.

### 4.1.30 V-HEL-VUL-030: Divided by zero #3

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | 119cbd0 |
| **Type** | Divide by Zero | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | collateralRate | | |

**Description** If the deployer forgets to call `spot.file()` on the token, then `mat` will be 0, causing a divide-by-zero error and therefore a revert.

```
1    function collateralRate(address token) external view returns (uint256) {
2        CollateralType memory collateralType = collaterals[token];
3        _checkIsLive(collateralType.live);
4        (,uint256 mat) = spotter.ilks(collateralType.ilk);
5        // (,,uint256 spot,,) = vat.ilks(collateralType.ilk);
6        // return spot / 10**9;
7        return 10 ** 45 / mat;
8    }
```

**Snippet 4.34:** Location where a divide-by-zero may occur in `collateralRate`

**Recommendation** The developers should require that `mat` is nonzero to provide a more helpful message so that the user knows why a revert occurred. Additionally, call any necessary methods like `spot.file` in methods such as the Interaction contract's `enableCollateralType` to avoid potential deployment errors.

### 4.1.31 V-HEL-VUL031: Unused Flop/Flap Code

| Severity | Warning | Commit | 119cbd0 |
|---|---|---|---|
| Type | Dead Code | Status | Fixed |
| Files | contracts/vow.sol | | |
| Functions | flop/flap/cage | | |

**Description** The Vow contract has code from MakerDAO that was intended to interact with the Flop/Flap contracts, which are not used by this project.

```
    function flop() external returns (uint id) {
        require(sump <= sub(sub(vat.sin(address(this)), Sin), Ash),
            "Vow/insufficient-debt");
        require(vat.usb(address(this)) == 0, "Vow/surplus-not-zero");
        Ash = add(Ash, sump);
        id = flopper.kick(address(this), dump, sump);
    }
    // Surplus auction or send surplus to multisig
    function flap() external returns (uint id) {
        if (lever != 0) {
            require(vat.usb(address(this)) >= add(add(vat.sin(address(this)), bump),
                hump), "Vow/insufficient-surplus");
            require(sub(sub(vat.sin(address(this)), Sin), Ash) == 0,
                "Vow/debt-not-zero");
            id = flapper.kick(bump, 0);
        } else {
            require(vat.usb(address(this)) >= add(vat.sin(address(this)), hump),
                "Vow/insufficient-surplus");
            require(sub(vat.sin(address(this)), Sin) == 0, "Vow/debt-not-zero");
            uint rad = sub(vat.usb(address(this)),add(vat.sin(address(this)), hump));
            vat.move(address(this), multisig, rad);
        }
    }

    function cage() external auth {
        require(live == 1, "Vow/not-live");
        live = 0;
        Sin = 0;
        Ash = 0;
        flapper.cage(vat.usb(address(flapper)));
        flopper.cage();
        vat.heal(min(vat.usb(address(this)), vat.sin(address(this))));
    }
```

**Snippet 4.35:** Location of the functions in the Vow contract that use the Flop/Flap contracts

**Recommendation** Since these contracts are not used in this project, these functions should be removed along with the FlopLike and FlapLike interfaces.

### 4.1.32 V-HEL-VUL032: usb.sol does not implement atomic allowance modification method

| Severity | Warning | Commit | c2b0aba |
|---|---|---|---|
| Type | Transaction Order | Status | Fixed |
| Files | contracts/usb.sol | | |
| Functions | N/A | | |

**Description** Due to well-known transaction reordering attacks on the ERC20 approve method, the developers should consider adding functions that atomically modify the `allowance` value.

**Recommendation** Consider adding the `increaseAllowance` and `decreaseAllowance` methods from the OpenZeppelin ERC20 implementation.

### 4.1.33 V-HEL-VUL-033: Interaction.borrow has ineffective use of mulDiv

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | c2b0aba |
| **Type** | Gas Optimization | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | borrow() | | |

**Description** Explicit use of `usbAmount * (10 ** 27)` in the "ceiling" check negates the benefit of using `hMath.mulDiv` to avoid uint256 multiplication overflow. Given that `usbAmount * RAY` is also computed and passed to `vat.move`, there is no benefit to using `hMath.mulDiv` over a simple multiply and divide.

```
1    function borrow(address token, uint256 usbAmount) external returns (uint256) {
2        ...
3
4        int256 dart = int256(hMath.mulDiv(usbAmount, 10 ** 27, rate));
5        if (uint256(dart) * rate < usbAmount * (10 ** 27)) {
6            dart += 1; //ceiling
7        }
8        vat.frob(collateralType.ilk, msg.sender, msg.sender, msg.sender, 0, dart);
9        uint256 mulResult = rate * uint256(dart);
10       vat.move(msg.sender, address(this), usbAmount * RAY);
11
12       ...
13   }
```

**Snippet 4.36:** Location were gas can be saved by using multiply/divide rather than `muldiv`

**Recommendation** Simply use multiply/divide to save on gas.

### 4.1.34 V-HEL-VUL-034: Constants for "Year" do not account for leap years

| Severity | Warning | Commit | c2b0aba |
|---|---|---|---|
| Type | Logical Error | Status | Fixed |
| Files | contracts/HelioRewards.sol, contracts/Interaction.sol | | |
| Functions | | | |

**Description** There are several constants of the form `YEAR = 365 * 24 * 3600` which represent the number of seconds in a year. However, on average, a year lasts 365.25 days. This leads to a slight rounding error where the duration will be off by 1 day every 4 years.

```
contract Interaction is Initializable, UUPSUpgradeable, OwnableUpgradeable {
    ...

    uint256 constant YEAR = 365 * 24 * 3600; //seconds

    ...
}
```

**Snippet 4.37:** Location of the `YEAR` constant in the Interaction contract

**Recommendation** For more precision, consider leap years in the computation.

### 4.1.35 V-HEL-VUL-035: Code inconsistent with comments

| Severity | Warning | Commit | c2b0aba |
|---|---|---|---|
| Type | Logical Error | Status | Fixed |
| Files | contracts/ceros/CerosRouter.sol | | |
| Functions | deposit() | | |

**Description** A comment in the `deposit` function is inconsistent with the computation of the local variable `poolABNBcAmount`. The comment states that `poolABNBcAmount` should be `amount - relayerFee - amount*(1-ratio)`, while the logic essentially sets the variable to `amount * ratio - relayerFee`.

```
1    function deposit() external payable override nonReentrant
2        returns (uint256 value)
3    {
4        ...
5
6        // let's calculate returned amount of aBNBc from BinancePool
7        // poolABNBcAmount = amount - relayerFee - amount*(1-ratio);
8        uint256 minumunStake = _pool.getMinimumStake();
9        uint256 relayerFee = _pool.getRelayerFee();
10       uint256 ratio = _certToken.ratio();
11       uint256 poolABNBcAmount;
12       if (amount >= minumunStake + relayerFee) {
13           poolABNBcAmount = ((amount - relayerFee) * ratio) / 1e18;
14       }
15
16       ...
17   }
```

**Snippet 4.38:** Location where a comment is inconsistent with the program logic in `deposit`

**Recommendation** Determine which value for `poolABNBcAmount` is correct and update the incorrect version of the computation.

### 4.1.36 V-HEL-VUL-036: Should only cast to interfaces that contracts inherit from

| Severity | Warning | Commit | c2b0aba |
|---|---|---|---|
| Type | Maintainability | Status | Fixed |
| Files | All | | |
| Functions | Ex. Interaction.totalPegLiquidity | | |

**Description** There are many instances where addresses are cast to interfaces that the underlying contract does not inherit from. This is common in MakerDAO as they typically define the interfaces required by a contract in its corresponding file. However, this pattern makes it difficult to maintain the source code since any updates to an interface will require updates to many locations. If one of these interfaces is incorrect, it could lead to a DOS or undefined behavior. Greater confidence is gained by only calling functions in inherited interfaces since the compiler will complain if the underlying implementation does not contain a concrete implementation of a function declared in the interface.

```
interface GemJoinLike {
    function join(address usr, uint256 wad) external;

    function exit(address usr, uint256 wad) external;

    function gem() external view returns (IERC20Upgradeable);
}
```

**Snippet 4.39:** The declaration of the GemJoinLike interface in Interaction.sol

```
contract GemJoin {
    ...
}
```

**Snippet 4.40:** The GemJoin contract declaration that does not inherit from GemJoinLike

**Recommendation** Update the contract definitions so that they inherit from the interfaces used by other contracts. In addition, only make calls to contracts using interfaces that the contract inherits from.

### 4.1.37 V-HEL-VUL-037: Two different versions of OpenZeppelin are being used

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | c2b0aba |
| **Type** | Maintainability | **Status** | Fixed |
| **Files** | All | | |
| **Functions** | N/A | | |

**Description** Currently two different versions of OpenZeppelin are in use. One is the vanilla OpenZeppelin library while the other is the upgradable OpenZeppelin library.

```
1 import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
2 import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
3 import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
4 import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.
      sol";
5 import "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";
```

**Snippet 4.41:** The OpenZeppelin imports in Interaction.sol

```
1 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
2 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

**Snippet 4.42:** The OpenZeppelin imports in AuctionProxy.sol

**Recommendation** Since the same contracts are cast to interfaces from both libraries, we suggest only using one of these libraries for consistency and in case of implementation differences.

### 4.1.38 V-HEL-VUL-038: Anyone can burn their own HAY stablecoin, affecting total supply

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | c2b0aba |
| **Type** | Logical Error | **Status** | Intended Behavior |
| **Files** | contracts/usb.sol | | |
| **Functions** | burn | | |

**Description** Currently anyone can burn their own HAY stablecoin, reducing the total supply. Without proper mitigation methods, giving users such an ability could affect the price of the stablecoin and therefore could cause HAY to unpeg from the target currency.

```
1   function burn(address usr, uint wad) external {
2       require(balanceOf[usr] >= wad, "Hay/insufficient-balance");
3       if (usr != msg.sender && allowance[usr][msg.sender] != type(uint256).max) {
4           require(allowance[usr][msg.sender] >= wad, "Hay/insufficient-allowance");
5           allowance[usr][msg.sender] = sub(allowance[usr][msg.sender], wad);
6       }
7       balanceOf[usr] = sub(balanceOf[usr], wad);
8       totalSupply    = sub(totalSupply, wad);
9       emit Transfer(usr, address(0), wad);
10  }
```

**Snippet 4.43:** The `burn` function of the HAY stablecoin

**Recommendation** The burning behavior of users should transfer the coins to an admin account. The admin can then be given the ability to truly burn coins to protect the stability of the currency.

**Developer Response** The developers acknowledged this risk and stated that they have other mechanisms to protect the stability of the HAY stablecoin.

### 4.1.39 V-HEL-VUL-039: owner variable shadowing by function parameter

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | c2b0aba |
| **Type** | Variable Shadowing | **Status** | Acknowledged |
| **Files** | contracts/ceros/{CeToken,CerosRouter,CeVault}.sol | | |
| **Functions** | depositABNBcFrom, _withdraw, _deposit, claimYieldsFor, etc. | | |

**Description** Although the affected contracts use the Ownable interface (or some derivative), they contain some functions that each declare `owner` as a function parameter. This can lead to issues where the `owner` function parameter is used rather than the `owner` contract variable. In addition, if the function parameter is renamed it can also cause the `owner` contract variable to be mistakenly used instead of the function parameter if the code isn't updated properly.

```
contract CeVault is IVault, OwnableUpgradeable, PausableUpgradeable,
    ReentrancyGuardUpgradeable
{
    ...

    function _deposit(
        address owner,
        address recipient,
        uint256 amount
    ) private returns (uint256) {
        uint256 ratio = _aBNBc.ratio();
        _aBNBc.transferFrom(msg.sender, address(this), amount);
        uint256 toMint = (amount * 1e18) / ratio;
        _depositors[owner] += amount; // aBNBc
        _ceTokenBalances[owner] += toMint;
        //  mint ceToken to recipient
        ICertToken(_ceToken).mint(recipient, toMint);
        emit Deposited(msg.sender, recipient, toMint);
        return toMint;
    }

    ...
}
```

**Snippet 4.44:** A function that shadows the `owner` contract variable

**Recommendation** To avoid confusion, never use owner as a function parameter name.

### 4.1.40 V-HEL-VUL-040: Invalid cast to ICertToken in CeVault.sol

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | c2b0aba |
| **Type** | Maintainability | **Status** | Acknowledged |
| **Files** | contracts/ceros/CeVault.sol | | |
| **Functions** | _deposit, _burn | | |

**Description** Based on the test cases provided by the developers, the `_ceToken` state variable is intended to be instantiated with a contract of type CeToken. However, the `_ceToken` state variable is incorrectly cast to ICertToken, which is not implemented by CeToken. Although the methods in CeVault currently do not invoke any ICertToken methods on `_ceToken` state variable, any attempted use of ICertToken methods will result in reverts.

```
1   function _deposit(
2       address owner,
3       address recipient,
4       uint256 amount
5   ) private returns (uint256) {
6       ...
7
8       ICertToken(_ceToken).mint(recipient, toMint);
9
10      ...
11  }
```

**Snippet 4.45:** Location where `_ceToken` is cast to an ICertToken

**Recommendation** The developers should change the type of `_ceToken` to IERC20 and update the `_ceToken` casts in `_deoosit` and `_burn`.

### 4.1.41 V-HEL-VUL-041: Dead Code

| | | | |
|---|---|---|---|
| **Severity** | Informational | **Commit** | c2b0aba |
| **Type** | Dead Code | **Status** | Fixed |
| **Files** | contracts/Interaction.sol | | |
| **Functions** | borrowApr | | |

**Description** There are currently two calls to `_checkIsLive(collateralType.live)` that immediately follow each other. Since this function is pure, the second call is redundant.

```
1    function borrowApr(address token) public view returns (uint256) {
2        CollateralType memory collateralType = collaterals[token];
3        _checkIsLive(collateralType.live);
4        _checkIsLive(collateralType.live);
5
6        ...
7    }
```

**Snippet 4.46:** Location of the dead code

**Recommendation** Remove one of the `_checkIsLive(collateralType.live)` calls.

### 4.1.42 V-HEL-VUL-042: ICertToken does not subclass IERC20

| Severity | Informational | Commit | c2b0aba |
|---|---|---|---|
| Type | Maintainability | Status | Acknowledged |
| Files | contracts/ceros/interfaces/ICertToken.sol | | |
| Functions | N/A | | |

**Description** Although ICertToken appears to be derived from IERC20, it does not inherit from IERC20. This puts it at risk of being inconsistent with the IERC20 interface.

```
interface ICertToken {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address to, uint256 amount) external returns (bool);
    function approve(address spender, uint256 amount) external returns (bool);
    function allowance(
        address owner,
        address spender
    ) external view returns (uint256);

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external returns (bool);

    function burn(address account, uint256 amount) external;
    function mint(address account, uint256 amount) external;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    function balanceWithRewardsOf(address account) external returns (uint256);
    function isRebasing() external returns (bool);
    function ratio() external view returns (uint256);
}
```

**Snippet 4.47:** The declaration of ICertToken

**Recommendation** Change the ICertToken contract so it inherits from IERC20.