Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Helio™

Veridise Inc.

June 17, 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 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	Fixed
Critical-Severity Issues	1	1
High-Severity Issues	6	4
Medium-Severity Issues	4	3
Low-Severity Issues	14	13
Warning-Severity Issues	16	6
Informational-Severity Issues	2	1
TOTAL	42	28

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 logic
- The Ceros deposit and withdraw mechanisms

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) - OR - 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 - OR - 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 - OR - 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

4 Vulnerability Report

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	Open
V-HEL-VUL-006	User Bypass of Functions	High	Fixed
V-HEL-VUL-007	No Token Check	High	Open
V-HEL-VUL-008	Test Suite Problem	Medium	Open
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	Open
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	Open
V-HEL-VUL-028	User Debt Error	Warning	Fixed
V-HEL-VUL-029	Address Desync	Warning	Open
V-HEL-VUL-030	Division by Zero #3	Warning	Open
V-HEL-VUL-031	Unused Flop/Flap Code	Warning	Open
V-HEL-VUL-032	Add Atomic Allowance Methods	Warning	Open
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	Open
V-HEL-VUL-036	Possible Interface Error	Warning	Open
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	Open
V-HEL-VUL-040	Incorrect State Variable Cast	Warning	Open
V-HEL-VUL-041	Dead Code	Informational	Fixed
V-HEL-VUL-042	Inconsistency with Interface	Informational	Open

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), "GemJoin/failed-
   transfer");
6      emit Join(usr, wad);
7  }
8
```

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

Recommendations 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 flash loan, 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 flash loan.


```
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

Recommendations 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:

- Because base is initially set to 0, and the rate calculated by jug.drip() depends on 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 be increased significantly after the first call.

For example, initializing base with 1000000003022266000000000 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 e.g. 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.

Recommendations 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 (frob in particular) to bypass Interaction.sol. Thus the state variables tracked in the Interaction contract (usersInDebt, 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 4.4: Location of the frob declaration in the Vat contract

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

Severity	High	Commit	c2b0aba
Type	Logical Error	Status	Open
Files	contracts/AuctionProxy.sol		
Functions	N/A		

Description Although AuctionProxy wraps the MakerDAO auction logic (clip), it only provides the methods startAuction and buyFromAuction . For the purposes of managing auctions, it would also be useful to provide methods to wrap the other auction functionalities, which include (but are not limited to):

- (Fixed) Reseting an auction.
- (Fixed) Checking whether an auction is still valid/needs to be reset
- (Fixed) Cancelling an auction (not planned).
- (Not Fixed) Call upchost in AuctionProxy.

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 a AuctionProxy.sol by instead interacting directly calling functions in the MakerDAO contract such as bark. These functions are currently marked as external with no access controls so anyone can call them. Thus, any interactions that developers want to enforce in AuctionProxy can be avoided by a user.

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

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

Description It is possible to call setCollateralType for an existing token, but with a different ilk, so that existing entry of collaterals will 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.

Recommendations : 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**	Open
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 workat 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.

Recommendations

- 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 Currently AuctionProxy passes on incentives given by MakerDAO as follows:

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

```

Snippet 4.5

Due to the integer division in vat.usb(address(this)) / RAY , any incentives that are less than RAY will be lost and not passed onto the user. These incentives that are < RAY will be given to other liquidators when they did not earn them. In addition, small rewards (< RAY) can result in 0 incentives being passed onto liquidators. Since these cases are not incentivized small loans may not be liquidated.

Possible Mitigation It seems like vat.move could be used to move the incentives to the user.

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 If a user is liquidated by startAuction , their Helio rewards are not updated. Consider the following scenario: a user deposits & borrows and then doesn't interact with the contract for a year. Then someone calls startAuction to (successfully) liquidate the user. Because helioRewards.drop() is never called for the user after the borrow, the user's unclaimed reward amount is never updated. Consequently, they will never get the Helio tokens that they were supposed to get.

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 a 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.

Recommendations 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.

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 method named `start()` emits an event named `Stop`, even though the contract has a `Start` event.

```
1 function start() public auth {
2     live = 1;
3     emit Stop(msg.sender);
4 }
```

Snippet 4.6: Location of the Stop emit in the start function

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

Severity	Low	**Commit**	e46d015
Type	Logical Error	**Status**	Open
Files	Interaction.sol		
Functions	buyFromAuction		

Description Interaction tracks a token's deposits and the users who are in debt. Both of these values can be impacted when a user's collateral is auction off; however, neither of these values are updated.

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 `provideInABNBc()` method 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, this unnecessarily exposes implementation details of the HelioProvider.

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`.

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 handling of usersInDebt in payback() suffers from the following issues:

- (Not Fixed) A user is removed from usersInDebt in payback when they have repaid all art in the corresponding urn. However this does not consider if a single user has multiple urns. If they do, they'll be removed from the set too early.
- (Fixed) The check for "no debt" is subject to rounding errors, which will prevent users from being removed from usersInDebt even after paying off all their debt:

```
if ((int256(rate * art) / 10 ** 27) == dart) {
```

It would be better to explicitly retrieve and check the urn amount. For example:

```
(, uint256 newArt) = vat.urns(collateralType.ilk, msg.sender);
if (newArt == 0) {
```

Update The developers have indicated that they will be removing the `_usersInDebt` functionality.

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 said that calling enableCollateralType on an existing token but with a different ilks is a feature. However, if ilk does not exist in the vat yet, then the ilk will not be initialized in the vat, so that future calls to deposit() and borrow() will revert when they call vat.frob() .
This can be worked around by a ward manually invoking vat.init(..) when using a new ilk; however, this is error-prone and should be handled directly by enableCollateralType()

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 In the affected methods, AuctionProxy assumes that it has sufficient permissions to the USB and/or USB join contract. If not, the affected methods will immediately revert. Note: while an admin can manually grant the permission, such a process is error-prone (e.g., what if an admin forgets? how about revoking permissions?).

In startAuction :

```
// vat.hope or vat.behalf required before this line
usbJoin.exit(address(this), vat.usb(address(this)) / RAY);
// should retract permission afterwards
```

In buyFromAuction :

```
// The following line results in vat usb transfer and USB burn.
// vat.hope/vat.behalf and usb.approve required before this line
usbJoin.join(address(this), usbMaxAmount);
// should retract permission & allowance afterwards
```

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 Interaction.buyFromAuction() requires the caller to pay HAY in order to receive collateral (via a transferFrom invoked by the auction proxy contract). Thus, the caller must 1) approve the auction proxy contract for the max amount rather than the Interaction contract; 2) invoke buyFromAction() ; and 3) unapprove the auction proxy contract.

If the caller is performing all of these actions in different transactions, then the caller may be vulnerable to transaction reordering attacks (esp. if usb.sol does not implement atomic allowance modification method is not addressed).

Some potential improvements include:

- Return the actual amount of HAY spent in the buyFromAuction method.
- Allow the user to approve the interaction contract instead, so that the user does not have to explicitly approve the auction proxy

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 buyFromAuction() calls `vat.hope()` on the collateral type's clip, but never calls `vat.nope()`. For security reasons, `vat.nope()` should be called when the clip no longer needs access to the auction proxy's balances. Furthermore, to limit access, it would be better to use `vat.behalf()` / `vat.regard()` instead of `vat.hope()` / `vat.hope()`, as the former grant more limited permissions.

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 function can revert if the amount parameter is set to -ink :

```
function estimatedLiquidationPrice(address token, address usr, int256 amount)
    external vie
w returns (uint256) {
    CollateralType memory collateralType = collaterals[token];
    _checkIsLive(collateralType.live);
    (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);
    }
    (, uint256 rate, uint256 spot,,) = vat.ilks(collateralType.ilk);
    (,uint256 mat) = spotter.ilks(collateralType.ilk);
    uint256 backedDebt = (art * rate / 10 ** 36) * mat;
    return backedDebt / ink;
}
```

Snippet 4.7

The revert occurs on the return line, since the computation ink = uint256(int256(ink) + amount) will cause ink to be 0.

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(address token, address usr) external view returns
       (uint25
2  6) {
3  CollateralType memory collateralType = collaterals[token];
4  _checkIsLive(collateralType.live);
5  (uint256 ink, uint256 art) = vat.urns(collateralType.ilk, usr);
6  (, uint256 rate,,,) = vat.ilks(collateralType.ilk);
7  (,uint256 mat) = spotter.ilks(collateralType.ilk);
8  uint256 backedDebt = (art * rate / 10 ** 36) * mat;
9  return backedDebt / ink;
10 }
```

Snippet 4.8

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 enableCollateralType() calls vat.rely(..) on the token's gem join. However, removeCollateralType does not call vat.deny(..) on the token's gem join. To improve security, vat.deny(...) should be added.

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 owner-only method 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 happens, there will be no error messages, reverts, or warnings.

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 owner-only initPool method multiple times on the same token, then the token will be duplicated in the poolsList array. Consequently, this means that rewards will be awarded multiple times for the same token due to the duplicate entries.

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 IERC20'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. The possibility that the approval is not set to zero could be more disastrous but it appears that all cases where this might lead to dangerous behavior are also guarded by _checkIsLive().

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.

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

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

Description Some combinations of multiplications & divisions directly use * and / , even when it is possible to use the hMath library to avoid overflow. Note that the hMath library takes more gas than raw multiplication & divide, so it may be better to leave the multiplies and divides as-is in some cases.

1. (Fixed) contracts/AuctionProxy.sol : in the `buyFromAuction` method,

```
uint256 hayMaxAmount = (maxPrice * collateralAmount) / RAY;
// replace with the following:
uint256 hayMaxAmount = FullMath.mulDiv(maxPrice, collateralAmount, RAY);
```

2. (Not Fixed) The `_deposit` and `_withdraw` functions in contracts/ceros/CeVault.sol are subject to possible overflow when computing toMint and realAmount respectively.

```
function _deposit(address account, uint256 amount)
private
returns (uint256)
{
    uint256 ratio = _aBNBc.ratio();
    _aBNBc.transferFrom(msg.sender, address(this), amount);
    uint256 toMint = (amount * 1e18) / ratio;
    ...
}

function _withdraw(
address owner,
address recipient,
uint256 amount
) private returns (uint256) {
    uint256 ratio = _aBNBc.ratio();
    uint256 realAmount = (amount * ratio) / 1e18;
    ...
}
```

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 Currently a user is added to the set usersInDebt upon a deposit. However at this point, the user has no debt. Instead it seems like the user should be added to this set in borrow.

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

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

Description Based on test/ceros/CerosRouter.js , a CerosRouter and a HelioProvider appear to maintain the property that they hold an address to each other (CerosRouter _provider and HelioProvider _ceRouter). In CerosRouter, it is possible to change the provider with the changeProvider method; however, there is no method in HelioProvider that allows the _ceRouter to be changed. This means that it is possible for the HelioProvider to point to the old router.

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

Severity	Warning	**Commit**	119cbd0
Type	Divide by Zero	**Status**	Open
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 and therefore a revert.

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

Recommendations

- Short term: require that mat is nonzero to provide a more helpful message, so that the user knows it is an issue with the contract.
- Long term: call any necessary methods like spot.file() in methods such as enableCollateralType() to avoid potential deployment errors.

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

Severity	Warning	**Commit**	119cbd0
Type	Dead Code	**Status**	Open
Files	contracts/vow.sol		
Functions	flop/flap/cage		

Description Vow.sol has code from MakerDAO that was intended to interact with the Flop/Flap contracts. 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	Open
Files	contracts/usb.sol		
Functions	N/A		

Description Due to well-known transaction reordering attacks on the ERC20 approve method, it would be helpful to add a method that atomically modifies the allowance/approval value. For example, consider adding the increaseAllovance and decreaseAllovance 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 there is also a usbAmount * RAY used for vat.move , there is no benefit to using hMath.mulDiv over a simple multiply and divide.

```
  int256 dart = int256(hMath.mulDiv(usbAmount, 10 ** 27, rate));
if (uint256(dart) * rate < usbAmount * (10 ** 27)) {
dart += 1; //ceiling
}
vat.frob(collateralType.ilk, msg.sender, msg.sender, msg.sender, 0, dart);
uint256 mulResult = rate * uint256(dart);
vat.move(msg.sender, address(this), usbAmount * RAY);
usbJoin.exit(msg.sender, usbAmount);
```

Note: the mulResult variable is also an unused variable.

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,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.

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

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

Description A comment in the deposit method is inconsistent with the calculation of the local variable poolABNBcAmount :

```
  // let's calculate returned amount of aBNBc from BinancePool
// poolABNBcAmount = amount - relayerFee - amount*(1-ratio);
uint256 minumunStake = _pool.getMinimumStake();
uint256 relayerFee = _pool.getRelayerFee();
uint256 ratio = _certToken.ratio();
uint256 poolABNBcAmount;
if (amount >= minumunStake + relayerFee) {
poolABNBcAmount = ((amount - relayerFee) * ratio) / 1e18;
}
```

The expression in the comment simplifies to:

```
amount - relayerFee - amount * (1 - ratio)
= amount - relayerFee - amount + amount * ratio
= amount * ratio - relayerFee
```

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

Severity	Warning	**Commit**	c2b0aba
Type	Maintainability	**Status**	Open
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 necessary interfaces in each 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.

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 vanilla openZeppelin while the other is the upgradable OpenZeppelin library. Since the same contracts are cast to interfaces from both libraries, we suggest only using one of these libraries for consistency in case there are 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 stable-coin and therefore could cause HAY to unpeg from the target currency.
Possible mitigation: Rather than burning, users could transfer their coin to an admin. The admin can then be given the ability to truly burn coins, affecting the total supply.

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

Severity	Warning	**Commit**	c2b0aba
Type	Variable Shadowing	**Status**	Open
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 may lead to subtle issues where the contract "owner" is mistakenly used instead of the address supplied as the function parameter (e.g., if the function parameter is renamed or has a typo).

Recommendation To avoid confusion, never use owner as a function parameter name unless it has to do with the Ownable interface.

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

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

Description Based on the test cases provided by the developers in test/ceros/CeRouter.test.js , the _ceToken state variable is 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 on _ceToken will result in reverts.

Recommendation

- change the type of _ceToken to IERC20 .
- remove the casts to ICertToken in _deposit 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        (uint256 duty,) = jug.ilks(collateralType.ilk);
7        uint256 principal = hMath.rpow((jug.base() + duty), YEAR, RAY);
8        return (principal - RAY) / (10 ** 7);
9    }
```

Snippet 4.9: Location of the dead code

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

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

Description Although ICertToken is meant to be derived from IERC20 , it does not subclass 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 4.10: The declaration of ICertToken