



Archimedes Finance

Smart Contract Security Audit

Prepared by: Halborn

Date of Engagement: October 31st, 2022 - November 11th, 2022

Visit: Halborn.com

DOCUMENT REVISION HISTORY	5
CONTACTS	5
1 EXECUTIVE OVERVIEW	6
1.1 INTRODUCTION	7
1.2 AUDIT SUMMARY	7
1.3 TEST APPROACH & METHODOLOGY	7
RISK METHODOLOGY	8
1.4 SCOPE	10
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	11
3 FINDINGS & TECH DETAILS	12
3.1 (HAL-01) HAL01 - OUSD BEHAVIOUR CAN LEAD TO UNDERFLOW WHEN OPENING A POSITION - MEDIUM	14
Description	14
Code Location	14
Proof Of Concept	16
Risk Level	16
Recommendation	16
Remediation Plan	16
3.2 (HAL-02) LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIAL- IZED CONTRACTS - LOW	17
Description	17
Code Location	17
Risk Level	18
Recommendation	18
Remediation Plan	18

3.3 (HAL-03) LACK OF PARAMETER PRECISION - LOW	19
Description	19
Code Location	19
Risk Level	19
Recommendation	20
Remediation Plan	20
3.4 (HAL-04) INCONSISTENT PARAMETER FORMATTING - LOW	21
Description	21
Code Location	21
Risk Level	21
Recommendation	22
Remediation Plan	22
3.5 (HAL-05) LACK OF 0 ADDRESS CHECK - INFORMATIONAL	23
Description	23
Code Location	23
Risk Level	23
Recommendation	24
Remediation Plan	24
3.6 (HAL-06) > 0 IS LESS EFFICIENT THAN != 0 FOR UINTS - INFORMATIONAL	25
Description	25
Code Location	25
Risk Level	25
Recommendation	26
Remediation Plan	26
3.7 (HAL-07) REDUNDANT VARIABLE USAGE - INFORMATIONAL	27
Description	27

Code Location	27
Risk Level	28
Recommendation	28
Remediation Plan	28
3.8 (HAL-08) REDUNDANT FUNCTIONS CALLS AND CHECKS - INFORMATIONAL	
29	
Description	29
Code Location	29
Risk Level	30
Recommendation	31
Remediation Plan	31
3.9 (HAL-09) USE OF POSTFIX OPERATORS RATHER THAN PREFIX OPERATORS - INFORMATIONAL	
32	
Description	32
Code Location	32
Risk Level	32
Recommendation	32
Remediation Plan	32
3.10 (HAL-10) FOR LOOP COULD BE REPLACE BY MULTIPLICATION - INFORMATIONAL	
33	
Description	33
Code Location	33
Risk Level	34
Recommendation	34
Remediation Plan	34
4 AUTOMATED TESTING	
4.1 STATIC ANALYSIS REPORT	
36	
Description	36

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/11/2022	Manuel García
0.2	Draft Review	11/14/2022	Kubilay Onur Gungor
0.3	Draft Review	11/14/2022	Gabi Urrutia
1.0	Remediation Plan	11/18/2022	Manuel García
1.1	Remediation Plan Review	11/21/2022	Kubilay Onur Gungor
1.2	Remediation Plan Review	11/21/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com
Manuel García	Halborn	Manuel.Diaz@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Archimedes Finance engaged Halborn to conduct a security audit on their finance protocol beginning on October 31st, 2022 and ending on November 11th, 2022. The security assessment was scoped to the smart contracts provided in the [Archimedes_Finance GitHub repository](#) [thisis-archimedes/Archimedes_Finance](#).

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned one full-time security engineer to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that all functions in the protocol smart contracts are intended.
- Identify potential security issues in Arcade bridge smart contracts.

In summary, Halborn identified many security risks that were mostly addressed by the [Archimedes Finance team](#).

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items

that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions ([solgraph](#))
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. ([MythX](#))
- Static Analysis of security for scoped contract, and imported functions. ([Slither](#))
- Testnet deployment ([Brownie](#), [Remix IDE](#))

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of **5** to **1** with **5** being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

- 5 - Almost certain an incident will occur.
- 4 - High probability of an incident occurring.
- 3 - Potential of a security incident in the long term.
- 2 - Low probability of an incident occurring.
- 1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

- 
- 5 - May cause devastating and unrecoverable impact or loss.
 - 4 - May cause a significant level of impact or loss.
 - 3 - May cause a partial impact or loss to many.
 - 2 - May cause temporary impact or loss.
 - 1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of **10** to **1** with **10** being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
----------	------	--------	-----	---------------

- 10** - CRITICAL
- 9** - **8** - HIGH
- 7** - **6** - MEDIUM
- 5** - **4** - LOW
- 3** - **1** - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The security assessment was scoped to the Solidity smart contracts of the audit branch

1. Solidity Smart Contracts

- (a) Repository: [Archimedes Finance](#)
- (b) Commit ID: [8df6cee5354480e5082c7cb753cf6d52504b616](#)
- (c) Fixed Commit ID: [dc38d4936eacff44b2ae1b1feece4d41dbd2748e](#)

Out-of-scope: External libraries and financial related attacks.

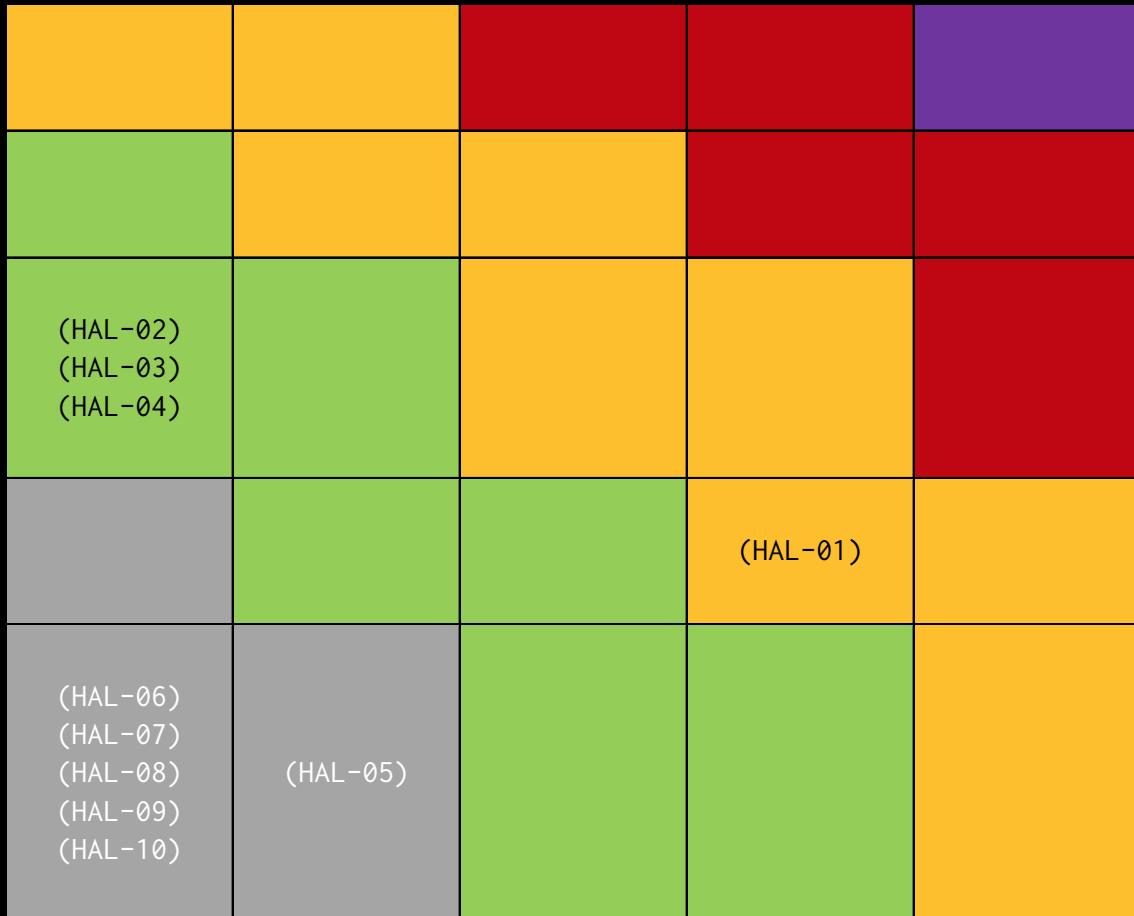
- The Proxy contracts that will be used by [Archimedes Fi team](#) to deploy the upgradeable contracts were not in the scope of this audit

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	6

LIKELIHOOD

IMPACT

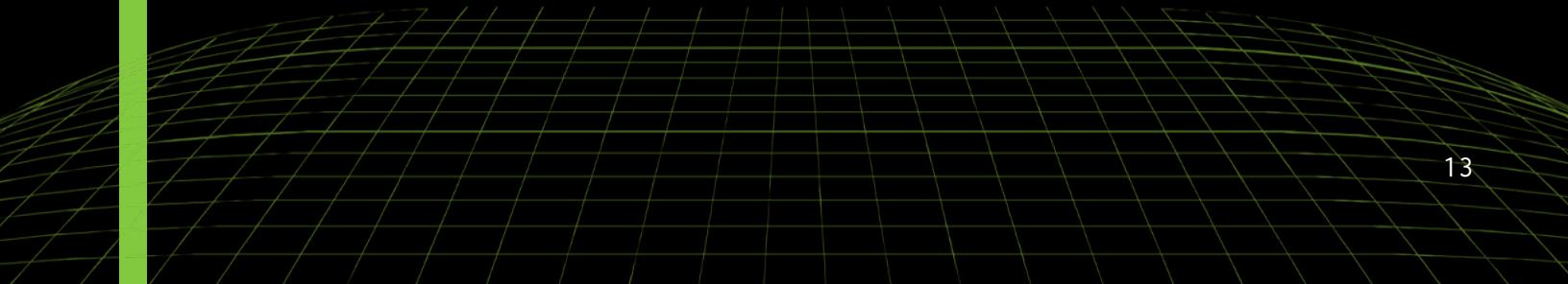


EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - LACK OF ALLOWANCE CHECK ON LEVERAGE ENGINE CAN LEAD TO OUSD UNDERFLOW	Medium	SOLVED - 11/18/2022
HAL02 - LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIALIZED CONTRACTS	Low	SOLVED - 11/18/2022
HAL03 - LACK OF PARAMETER PRECISION	Low	NOT SOLVED
HAL04 - INCONSISTENT PARAMETER FORMATTING	Low	RISK ACCEPTED
HAL05 - LACK OF 0 ADDRESS CHECK	Informational	SOLVED - 11/18/2022
HAL06 - > 0 IS LESS EFFICIENT THAN != 0 FOR UINTS	Informational	SOLVED - 11/18/2022
HAL07 - REDUNDANT VARIABLE USAGE	Informational	PARTIALLY SOLVED
HAL08 - REDUNDANT FUNCTION CALLS AND CHECKS	Informational	PARTIALLY SOLVED
HAL09 - USE OF POSTFIX OPERATORS RATHER THAN PREFIX OPERATORS	Informational	SOLVED - 11/18/2022
HAL10 - FOR LOOP COULD BE REPLACED BY MULTIPLICATION	Informational	ACKNOWLEDGED



FINDINGS & TECH DETAILS



3.1 (HAL-01) HAL01 - OUSD BEHAVIOUR CAN LEAD TO UNDERFLOW WHEN OPENING A POSITION - MEDIUM

Description:

It has been observed that once a user tries to create any position, the Leverage Engine will try to transfer the `OUSD` principle to the Leverage Engine. However, due to `OUSD` behavior, trying to transfer more tokens than allowance will revert with an Underflow error message. Although in recent versions of solidity, this is no longer a critical issue as the value doesn't reset to the maximum value, it is still recommended to verify that the user has enough allowance before transferring the tokens, as the underflow error can be confusing to users.

Code Location:

```
Listing 1: contracts/LeverageEngine.sol (Line 99)

62 /* Non-privileged functions */
63
64 /// @dev deposit OUSD under NFT ID
65 ///
66 /// User sends OUSD to the contract.
67 /// We mint NFT, assign to msg.sender and do the leverage cycles
68 ///
69 /// @param ousdPrinciple the amount of OUSD sent to Archimedes
70 /// @param cycles How many leverage cycles to do
71 /// @param maxArchAmount max amount of Arch tokens to burn for
↳ position
72 function createLeveragedPosition(
73     uint256 ousdPrinciple,
74     uint256 cycles,
75     uint256 maxArchAmount
76 ) external nonReentrant whenNotPaused returns (uint256) {
77     // add some minor buffer to the arch we will use for the
↳ position
78     if (cycles == 0 || cycles > _parameterStore.
↳ getMaxNumberOfCycles()) {
```

```
79         revert("Invalid number of cycles");
80     }
81     // console.log("ousdPrinciple %s", ousdPrinciple);
82     if (ousdPrinciple < _parameterStore.getMinPositionCollateral()
↳ ) {
83         revert("Collateral lower then min");
84     }
85     // console.log("maxArchAmountBufferedDown %s", maxArchAmount);
86     uint256 maxArchAmountBufferedDown = maxArchAmount;
87     uint256 lvUSDAmount = _parameterStore.
↳ getAllowedLeverageForPositionWithArch(ousdPrinciple, cycles,
↳ maxArchAmountBufferedDown);
88     uint256 lvUSDAmountNeedForArguments = _parameterStore.
↳ getAllowedLeverageForPosition(ousdPrinciple, cycles);
89     /// check that user gave enough arch allowance for cycle-
↳ principle combo
90     require(lvUSDAmountNeedForArguments - 1 <= lvUSDAmount, "cant
↳ get enough lvUSD");
91     uint256 archNeededToBurn = (_parameterStore.
↳ calculateArchNeededForLeverage(lvUSDAmount) / 10000) * 10000; //
↳ minus 1000 wei
92     // console.log("archNeededToBurn %s", maxArchAmount);
93
94     require(archNeededToBurn <= maxArchAmountBufferedDown, "Not
↳ enough Arch given for Pos");
95     uint256 availableLev = _coordinator.getAvailableLeverage();
96     require(availableLev >= lvUSDAmount, "Not enough available
↳ lvUSD");
97     _burnArchTokenForPosition(msg.sender, archNeededToBurn);
98     uint256 positionTokenId = _positionToken.safeMint(msg.sender);
99     _ousd.safeTransferFrom(msg.sender, _addressCoordinator,
↳ ousdPrinciple);
100    _coordinator.depositCollateralUnderNFT(positionTokenId,
↳ ousdPrinciple);
101    _coordinator.getLeveragedOUSD(positionTokenId, lvUSDAmount);
102    uint256 psoitionExpireTime = _coordinator.
↳ getPositionExpireTime(positionTokenId);
103
104    emit PositionCreated(msg.sender, positionTokenId,
↳ ousdPrinciple, lvUSDAmount, archNeededToBurn, psoitionExpireTime);
105
106    return positionTokenId;
107 }
```

Proof Of Concept:

- The user calls the `createLeveragedPosition()` in the `LeverageEngine` contract having enough OUSD balance but not enough allowance.
- Transaction revert due to integer underflow.

Risk Level:

Likelihood - 4

Impact - 2

Recommendation:

Implementing a require statement validating that the user's allowance for the Leverage Engine is equal to or greater than the OUSD principle is recommended.

Remediation Plan:

SOLVED: The Archimedes team fixed the issue by checking that the OUSD allowance is equal to or greater than the OUSD principle provided in the arguments.

3.2 (HAL-02) LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIALIZED CONTRACTS - LOW

Description:

Multiple contracts are using the `Initializable` module from OpenZeppelin. To prevent leaving an implementation contract uninitialized [OpenZeppelin's documentation](#) recommends adding the `_disableInitializers` function in the constructor to lock the contracts automatically when they are deployed:

Code Location:

Listing 2

```
1 /**
2  * @dev Locks the contract, preventing any future reinitialization
3  * . This cannot be part of an initializer call.
4  * Calling this in the constructor of a contract will prevent that
5  * contract from being initialized or reinitialized
6  * to any version. It is recommended to use this to lock
7  * implementation contracts that are designed to be called
8  * through proxies.
9 */
10 function _disableInitializers() internal virtual {
11     require(!_initializing, "Initializable: contract is
12     initializing");
13     if (_initialized < type(uint8).max) {
14         _initialized = type(uint8).max;
15         emit Initialized(type(uint8).max);
16     }
17 }
```

Risk Level:

Likelihood - 1

Impact - 3

Recommendation:

Consider calling the `_disableInitializers` function in the `TimeLockControllerUpgradeable` contract constructor as well:

Listing 3

```
1 /// @custom:oz-upgrades-unsafe-allow constructor
2 constructor() {
3     _disableInitializers();
4 }
```

Remediation Plan:

SOLVED: The Archimedes team fixed the issue by calling the `_disableInitializers` function in the contract constructor.

3.3 (HAL-03) LACK OF PARAMETER PRECISION - LOW

Description:

It has been detected that some parameters used in the different contracts are set and used in a way that lets the minimum settable value at 1%.

Code Location:

```
Listing 4: ParameterStore.sol (Line 40)

34     _maxNumberOfCycles = 10;
35     _originationFeeRate = 5 ether / 100;
36     _globalCollateralRate = 90;
37     _rebaseFeeRate = 10 ether / 100; // meaning 10%
38     _treasuryAddress;
39     _curveGuardPercentage = 90;
40     _slippage = 2; // 2%
41     _archToLevRatio = 1 ether; // meaning 1 arch is equal 1
↳ lvUSD
42     _curveMaxExchangeGuard = 50; // meaning we allow exchange
↳ with get 50% more then we expected
43     _treasuryAddress = address(0);
```

For example, `_slippage` is set to `2`, implying that a maximum slippage of 2% is allowed. However, if this value needs to be modified in the future or if user-controlled slippage is implemented, it would only accept integer values, such as `0%`, `1%`, `2%`, etc.

Risk Level:

Likelihood - 1

Impact - 3

FINDINGS & TECH DETAILS

Recommendation:

Consider increasing the digits in the % calculations to allow more flexibility when using these parameters.

Remediation Plan:

NOT SOLVED: The Archimedes team did not solve the issue.

3.4 (HAL-04) INCONSISTENT PARAMETER FORMATTING - LOW

Description:

`ParameterStore.sol` contract stores global parameters used when managing positions, such as `_maxNumberOfCycles`, `_originationFeeRate`, etc.

However, it has been detected that parameters with a similar format or that are used in similar ways (like percentages, for example) are defined and used differently, which could cause unexpected contract behavior (or even worse situations such as fund loss) if they are mistakenly modified.

Code Location:

```
Listing 5: ParameterStore.sol (Lines 36,37)

34     _maxNumberOfCycles = 10;
35     _originationFeeRate = 5 ether / 100;
36     _globalCollateralRate = 90;
37     _rebaseFeeRate = 10 ether / 100; // meaning 10%
38     _treasuryAddress;
39     _curveGuardPercentage = 90;
40     _slippage = 2; // 2%;
41     _archToLevRatio = 1 ether; // meaning 1 arch is equal 1
↳ 1vUSD
42     _curveMaxExchangeGuard = 50; // meaning we allow exchange
↳ with get 50% more then we expected
43     _treasuryAddress = address(0);
```

In this example can be seen how two different rates (with 90% and 10% values, respectively) are calculated differently.

Risk Level:

Likelihood - 1

Impact - 3

FINDINGS & TECH DETAILS

Recommendation:

It is strongly recommended to unify the parameter criteria in a way that allows contract administrators to set parameters of the same nature in the same way.

Remediation Plan:

RISK ACCEPTED: The Archimedes team accepted the risk of this finding.

3.5 (HAL-05) LACK OF 0 ADDRESS CHECK - INFORMATIONAL

Description:

When setting the dependencies for the contract through the `setDependencies()` functions, the 0 address is not being checked in any of the contracts.

Code Location:

```
Listing 6: contracts/LeverageEngine.sol

43 function setDependencies(
44     address addressCoordinator,
45     address addressPositionToken,
46     address addressParameterStore,
47     address addressArchToken,
48     address addressOUSD
49 ) external nonReentrant onlyAdmin {
50     _addressCoordinator = addressCoordinator;
51     _coordinator = ICoordinator(addressCoordinator);
52     _addressPositionToken = addressPositionToken;
53     _positionToken = PositionToken(addressPositionToken);
54     _addressParameterStore = addressParameterStore;
55     _parameterStore = ParameterStore(addressParameterStore);
56     _addressArchToken = addressArchToken;
57     _archToken = ArchToken(addressArchToken);
58     _addressOUSD = addressOUSD;
59     _ousd = IERC20Upgradeable(_addressOUSD);
60 }
61
```

Risk Level:

Likelihood - 2

Impact - 1

FINDINGS & TECH DETAILS

Recommendation:

When setting an address variable, always make sure the value is not zero.

Remediation Plan:

SOLVED: The Archimedes team fixed the issue by adding a zero address check in the `setDependencies()` functions.

3.6 (HAL-06) > 0 IS LESS EFFICIENT THAN $\neq 0$ FOR UINTS - INFORMATIONAL

Description:

The use of `>` consumes more gas than `\neq` . There are some cases where both can be used indistinctly, such as in unsigned integers where numbers can't be negative, and as such, there is only a need to check that a number is not 0.

Code Location:

`CDPosition.sol`

- Line 40: `require(_nftCDP[nftID].oUSDPrinciple > 0, "NFT ID must exist");`

`Coordinator.sol`

- Line 143: `require(numberOfSharesInPosition > 0, "Position has no shares");`

`ParameterStore.sol`

- Line 63: `require(newMaxNumberOfCycles < 20 && newMaxNumberOfCycles > 0, "New max n of cycles out of range");`
- Line 81: `require(newGlobalCollateralRate <= 100 && newGlobalCollateralRate > 0, "New collateral rate out of range");`
- Line 87: `require(newMaxNumberOfCycles < 20 && newMaxNumberOfCycles > 0, "New max n of cycles out of range");`

Risk Level:

Likelihood - 1

Impact - 1

FINDINGS & TECH DETAILS

Recommendation:

Use != instead of > in cases where both can be used.

Remediation Plan:

SOLVED: The Archimedes team fixed the issue by replacing > with !=.

3.7 (HAL-07) REDUNDANT VARIABLE USAGE - INFORMATIONAL

Description:

Redundant variable assignments have been found through the code. These statements increase gas costs for contract deploying and interactions, and impact code readability.

Code Location:

In this example, `maxArchAmountBufferedDown` is assigned with the value from the `maxArchAmount` parameter instead of using the parameter variable directly.

Listing 7: contracts/LeverageEngine.sol (Line 72)

```
72 function createLeveragedPosition(
73     uint256 ousdPrinciple,
74     uint256 cycles,
75     uint256 maxArchAmount
76 ) external nonReentrant whenNotPaused returns (uint256) {
77     // add some minor buffer to the arch we will use for the
    ↳ position
78     if (cycles == 0 || cycles > _parameterStore.
    ↳ getMaxNumberOfCycles()) {
79         revert("Invalid number of cycles");
80     }
81     // console.log("ousdPrinciple %s", ousdPrinciple);
82     if (ousdPrinciple < _parameterStore.getMinPositionCollateral()
    ↳ ) {
83         revert("Collateral lower than min");
84     }
85     // console.log("maxArchAmountBufferedDown %s", maxArchAmount);
86     uint256 maxArchAmountBufferedDown = maxArchAmount;
```

Moreover, in this other example `cyclePrinciple` is assigned with the value from the `principle` parameter.

Listing 8: contracts/ParameterStore.sol (Line 171)

```
171 function getAllowableLeverageForPosition(uint256 principle, uint256
172   ↳ numberofCycles) public view returns (uint256) {
173     require(numberofCycles <= _maxNumberOfCycles, "Cycles greater
174   ↳ than max allowed");
173     uint256 leverageAmount = 0;
174     uint256 cyclePrinciple = principle;
```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

Redundant variable assignments should be deleted to improve code readability and usability and reduce gas costs when deploying or interacting with the contract.

Remediation Plan:

PARTIALLY SOLVED: The [Archimedes](#) team fixed the issue in some instances, but it has been partially solved.

3.8 (HAL-08) REDUNDANT FUNCTIONS CALLS AND CHECKS - INFORMATIONAL

Description:

While requirements are needed for proper usage and security of the contracts, some redundant validations have been seen.

Code Location:

In this instance, the `getAllowedLeverageForPositionWithArch` function will already call and return the `getAllowedLeverageForPosition` function return value. Moreover, this function will already revert if not enough ARCH is provided. Making the require statements in lines 90 and 94 redundant.

Listing 9: contracts/LeverageEngine.sol (Lines 87,88,90,94)

```

72 function createLeveragedPosition(
73     uint256 ousdPrinciple,
74     uint256 cycles,
75     uint256 maxArchAmount
76 ) external nonReentrant whenNotPaused returns (uint256) {
77     // add some minor buffer to the arch we will use for the
↳ position
78     if (cycles == 0 || cycles > _parameterStore.
↳ getMaxNumberOfCycles()) {
79         revert("Invalid number of cycles");
80     }
81     // console.log("ousdPrinciple %s", ousdPrinciple);
82     if (ousdPrinciple < _parameterStore.
↳ getMinPositionCollateral()) {
83         revert("Collateral lower than min");
84     }
85     // console.log("maxArchAmountBufferedDown %s",
↳ maxArchAmount);
86     uint256 maxArchAmountBufferedDown = maxArchAmount;
87     uint256 lvUSDAmount = _parameterStore.
↳ getAllowedLeverageForPositionWithArch(ousdPrinciple, cycles,
↳ maxArchAmountBufferedDown);

```

```
88         uint256 lvUSDAmountNeedForArguments = _parameterStore.
89         getAllowedLeverageForPosition(ousdPrinciple, cycles);
89         /// check that user gave enough arch allowance for cycle-
90         principle combo
90         require(lvUSDAmountNeedForArguments - 1 <= lvUSDAmount, "
90         cant get enough lvUSD");
91         uint256 archNeededToBurn = (_parameterStore.
91         calculateArchNeededForLeverage(lvUSDAmount) / 10000) * 10000; //
91         minus 1000 wei
92         // console.log("archNeededToBurn %s", maxArchAmount);
93
94         require(archNeededToBurn <= maxArchAmountBufferedDown, "
94         Not enough Arch given for Pos");
95         uint256 availableLev = _coordinator.getAvailableLeverage()
95         ;
96         require(availableLev >= lvUSDAmount, "Not enough available
96         lvUSD");
97         _burnArchTokenForPosition(msg.sender, archNeededToBurn);
98         uint256 positionTokenId = _positionToken.safeMint(msg.
98         sender);
99         _ousd.safeTransferFrom(msg.sender, _addressCoordinator,
99         ousdPrinciple);
100         _coordinator.depositCollateralUnderNFT(positionTokenId,
100         ousdPrinciple);
101         _coordinator.getLeveragedOUSD(positionTokenId, lvUSDAmount
101         );
102         uint256 psoitionExpireTime = _coordinator.
102         getPositionExpireTime(positionTokenId);
103
104         emit PositionCreated(msg.sender, positionTokenId,
104         ousdPrinciple, lvUSDAmount, archNeededToBurn, psoitionExpireTime);
105
106         return positionTokenId;
107     }
```

Risk Level:

Likelihood - 1

Impact - 1

FINDINGS & TECH DETAILS

Recommendation:

Avoid calling the same function twice unnecessarily.

Remediation Plan:

PARTIALLY SOLVED: The Archimedes team fixed the issue in some instances, but it has been partially solved.

3.9 (HAL-09) USE OF POSTFIX OPERATORS RATHER THAN PREFIX OPERATORS - INFORMATIONAL

Description:

The use of postfix operators `i++` consume more gas than prefix operators `++i`.

Code Location:

`ParameterStore.sol`

- Line 176: `for (uint256 i = 0; i < number0fCycles; ++i){`

`PositionToken.sol`

- Line 88: `for (uint256 i = 0; i < _addressToTokensOwnedMapping[from].length; i++){`

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to use prefix operators rather than postfix.

Remediation Plan:

SOLVED: The `Archimedes team` fixed the issue.

3.10 (HAL-10) FOR LOOP COULD BE REPLACE BY MULTIPLICATION - INFORMATIONAL

Description:

For loops are especially useful in functions where some complex logic is involved in each loop. However, in cases where the for loop will only add a constant value to a variable, it is recommended to use multiplication instead, as it will save gas.

Code Location:

For example, the loop in the `getAllowedLeverageForPosition` function:

```
Listing 10: contracts/ParameterStore.sol (Line 179)

171 function getAllowedLeverageForPosition(uint256 principle, uint256
172     numberOFCycles) public view returns (uint256) {
173     require(numberOFCycles <= _maxNumberOFCycles, "Cycles greater
174     than max allowed");
175     uint256 leverageAmount = 0;
176     uint256 cyclePrinciple = principle;
177     // console.log("getAllowedLeverageForPosition principle %s,
178     // numberOFCycles %s", principle / 1 ether, numberOFCycles);
179     for (uint256 i = 0; i < numberOFCycles; ++i) {
180         // console.log("getAllowedLeverageForPosition looping on
181         // cycles");
182         cyclePrinciple = (cyclePrinciple * _globalCollateralRate)
183         / 100;
184         leverageAmount += cyclePrinciple;
185     }
186     // console.log("getAllowedLeverageForPosition: leverageAmount
187     // %s", leverageAmount / 1 ether);
188     return leverageAmount;
189 }
```

FINDINGS & TECH DETAILS

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to replace this loop by a multiplication.

Remediation Plan:

ACKNOWLEDGE: The Archimedes team acknowledged the issue.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither Results:

- AccessController.sol

AUTOMATED TESTING

- ArchToken .so

AUTOMATED TESTING

- BasicAccessController.sol

- CDPosition.sol

AUTOMATED TESTING

- Coordinator, sol

AUTOMATED TESTING

AUTOMATED TESTING

- Exchanger.sol

AUTOMATED TESTING

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used>

```
AccessControl,_setRoleAdmin(bytes32,bytes32) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/access/AccessControl.sol@214-218) is never used and should be removed
AccessControl,_setupRole(bytes32,address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/access/AccessControl.sol@205-207) is never used and should be removed
AccessControlUpgradeable,_setRoleAdmin(bytes32,bytes32) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/access/AccessControlUpgradeable.sol@220-224) is never used and should be removed
AccessControlUpgradeable,_setupRole(bytes32,address) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/access/AccessControlUpgradeable.sol@211-213) is never used and should be removed
AddressUpgradeable,functionCall(address,bytes) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/AddressUpgradeable.sol@55-87) is never used and should be removed
AddressUpgradeable,functionStaticCall(address,bytes) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/AddressUpgradeable.sol@147-149) is never used and should be removed
AddressUpgradeable,sendValue(address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/AddressUpgradeable.sol@157-166) is never used and should be removed
AddressUpgradeable,sendValue(address,uint256,int) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/AddressUpgradeable.sol@68-65) is never used and should be removed
Arcto,requireAdmin(bytes32) (.brownie/packages/arcto/arcto@0.1.0/contracts/arcto/Arcto.sol@11-12) is never used and should be removed
BasicAccessController,_requireAdmin() (.jobs/arctos/brownie/contracts/basicAccessController.sol@67-69) is never used and should be removed
Context,_msgData() (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/utils/context/_msgData.sol@21-23) is never used and should be removed
Context,_textEncoder() (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/utils/context/_textEncoder.sol@19-20) is never used and should be removed
ContextUpgradeable,_ContextInit(unchecked) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/context/_ContextUpgradeable.sol@21-22) is never used and should be removed
Counters,decrement(Counters.Counter) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/units/Counters.sol@32-38) is never used and should be removed
CountersUpgradeable,decrement(CountersUpgradeable.Counter) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/CountersUpgradeable.sol@40-42) is never used and should be removed
CountersUpgradeable,reset(CountersUpgradeable.Counter) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/CountersUpgradeable.sol@43-45) is never used and should be removed
CountersUpgradeable,revert(CountersUpgradeable.Counter) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/CountersUpgradeable.sol@46-48) is never used and should be removed
ECDSA,recoverBytes32(bytes) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/cryptography/ECDSA.sol@104-106) is never used and should be removed
ECDSA,toEthSignedMessageHash(bytes) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/cryptography/ECDSA.sol@202-206) is never used and should be removed
ECDSA,toEthSignedMessageHash(bytes2) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/cryptography/ECDSA.sol@207-210) is never used and should be removed
ECDSA,toEthSignedMessageHash(bytes32) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/cryptography/ECDSA.sol@211-214) is never used and should be removed
ERC165,tryRecoverBytes32(bytes) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol@24-25) is never used and should be removed
ERC165Upgradeable,.ERC165_init() (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol@26-27) is never used and should be removed
ERC165Upgradeable,.ERC165_initUnchained() (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol@28-29) is never used and should be removed
ERC165Upgradeable,.ERC165Upgradeable_init() (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol@30-32) is never used and should be removed
ERC165Upgradeable,.changeAdmin(address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol@139-142) is never used and should be removed
ERC1967Upgradeable,.setAdmin(address) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/introspection/ERC1967UpgradeableUpgradeable.sol@120-124) is never used and should be removed
ERC20,.add(uint256,address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20/ERC20.sol@208-209) is never used and should be removed
ERC20Votes,.add(uint256,address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20Votes/ERC20Votes.sol@220-244) is never used and should be removed
ERC20Votes,.sub(uint256,address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20Votes/ERC20Votes.sol@197-210) is never used and should be removed
ERC20Votes,.subr(uint256,address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20Votes/ERC20Votes.sol@211-214) is never used and should be removed
ERC721BurnableUpgradeable,.ERC721Burnable_init() (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/token/ERC721/extensions/ERC721BurnableUpgradeable.sol@10-19) is never used and should be removed
ERC721BurnableUpgradeable,.setAdmin(address) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/token/ERC721/extensions/ERC721BurnableUpgradeable.sol@129-132) is never used and should be removed
ERC721BurnableUpgradeable,.setBeaconAddress() (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeableUpgradeable.sol@165-172) is never used and should be removed
ERC1967UpgradeableUpgradeable,.upgradeBeaconToAndCall(address,bytes,bool) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeableUpgradeable.sol@180-190) is never used and should be removed
ERC20,.burn(address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20/ERC20.sol@208-209) is never used and should be removed
ERC20Votes,.add(uint256,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20Votes/ERC20Votes.sol@220-244) is never used and should be removed
ERC20Votes,.sub(uint256,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20Votes/ERC20Votes.sol@197-210) is never used and should be removed
ERC20Votes,.subr(uint256,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20Votes/ERC20Votes.sol@211-214) is never used and should be removed
ERC20Votes,.tryTransferFrom(address,address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/token/ERC20/ERC20.sol@149-151) is never used and should be removed
SafeCast,toInt168(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@204-207) is never used and should be removed
SafeCast,toInt176(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@187-190) is never used and should be removed
SafeCast,toInt192(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@153-156) is never used and should be removed
SafeCast,toInt200(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@136-139) is never used and should be removed
SafeCast,toInt208(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@119-120) is never used and should be removed
SafeCast,toInt216(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@102-103) is never used and should be removed
SafeCast,toInt224(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@510-513) is never used and should be removed
SafeCast,toInt232(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@468-471) is never used and should be removed
SafeCast,toInt241(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@410-413) is never used and should be removed
SafeCast,toInt248(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@374-377) is never used and should be removed
SafeCast,toInt256(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@558-561) is never used and should be removed
SafeCast,toInt401(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@476-479) is never used and should be removed
SafeCast,toInt56(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@442-445) is never used and should be removed
SafeCast,toInt64(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@425-428) is never used and should be removed
SafeCast,toInt721(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@406-409) is never used and should be removed
SafeCast,toInt80(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@391-394) is never used and should be removed
SafeCast,toInt88(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@374-377) is never used and should be removed
SafeCast,toInt96(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts/math/SafeCast.sol@357-360) is never used and should be removed
SafeERC20Upgradeable,.safeApprove(IERC20Upgradeable,address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/token/ERC20/ERC20.sol@46-59) is never used and should be removed
SafeERC20Upgradeable,.safeBacrossAllowanceIERC20Upgradeable(address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/token/ERC20/ERC20.sol@70-81) is never used and should be removed
SafeERC20Upgradeable,.safeIncreaseAllowanceIERC20Upgradeable(address,address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/token/ERC20/ERC20.sol@61-68) is never used and should be removed
SafeERC20Upgradeable,.safePermitIERC20PermitUpgradeable(address,address,uint256,int256,int256,uint8,bytes32) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/token/ERC20/ERC20.sol@22-28) is never used and should be removed
SafeERC20Upgradeable,.safeTransferIERC20Upgradeable(address,address,uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/token/ERC20/ERC20.sol@22-28) is never used and should be removed
StorageSlot,_getStorageByIndex(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-storage/StorageSlot.sol@77-78) is never used and should be removed
StorageSlot,_getStorageByIndex(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-storage/StorageSlot.sol@92-97) is never used and should be removed
Strings,toEIP55String(address) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-utils/Strings.sol@72-74) is never used and should be removed
Strings,toHexString(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-utils/Strings.sol@141-152) is never used and should be removed
Strings,toString(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-utils/Strings.sol@16-19) is never used and should be removed
StringsUpgradeable,toHexString(uint256) (.brownie/packages/OpenZeppelin/openzeppelin-upgradeable@4.7.0/contracts-upgradeable/utils/StringsUpgradeable.sol@441-521) is never used and should be removed
UIUPUpgradeable,.IUPUpgradeableInit(unchained) (.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.7.0/contracts-upgradeable/proxy/utils/UIUPUpgradeable.sol@26-27) is never used and should be removed
```


- ParameterStore.sol

- PoolManager.sol

- PositionToken.sol

AUTOMATED TESTING

- VaultOUSD.sol

AUTOMATED TESTING

Issues found by slither are either reported above or false positives.

THANK YOU FOR CHOOSING
HALBORN