Archimedes Finance - Auctions

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 19th, 2022 - December 25th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/19/2022	Francisco González
0.2	Document Updates	12/22/2022	Francisco González
0.3	Draft Review	12/22/2022	Roberto Reigada
0.4	Draft Review	12/23/2022	Piotr Cielas
0.5	Draft Review	12/23/2022	Gabi Urrutia
1.0	Remediation Plan	01/10/2023	Francisco González
1.1	Remediation Plan Review	01/11/2023	Roberto Reigada
1.2	Remediation Plan Review	01/11/2023	Piotr Cielas
1.3	Remediation Plan Review	01/11/2023	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com
Francisco González	Halborn	Francisco.Villarejo@halborn.com

EXECUTIVE OVERVIEW

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Archimedes Finance is an experimental lending and borrowing platform built on top of AMMs such as Curve.

Archimedes Finance engaged Halborn to conduct a security audit on their smart contracts beginning on December 19th, 2022 and ending on December 25th, 2022 . The security assessment was scoped to the smart contracts and functions detailed in the Scope section of this report, along with Commit hashes and further details.

1.2 AUDIT SUMMARY

The team at Halborn was provided 1 week for the engagement and assigned 1 full-time security engineer to audit the security of the programs in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audits is to:

- Identify potential security issues within the programs
- Ensure that smart contract functions operate as intended

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which have been addressed and acknowledged by Archimedes Finance . The main ones are the following:

- Ensure that there is an auction running before releasing new leverage.
- Validate leverage obtained from closed positions before releasing it.
- Implement a front-running protection mechanism to prevent users from cheating in auctions.

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 code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during 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 hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE, Ganache, Foundry)

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

Code repositories:

1. BNPL

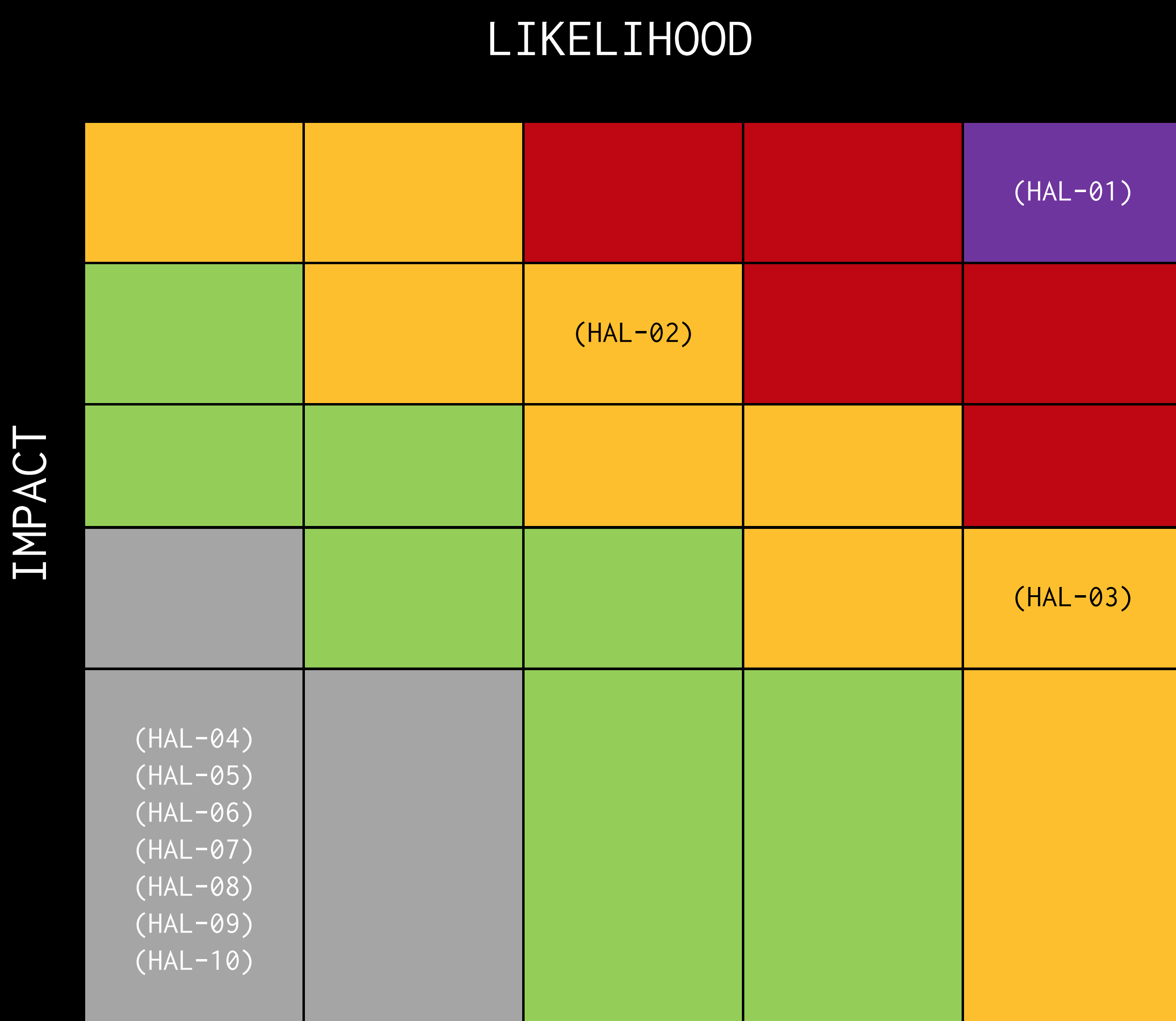
- Repository: hisisarchimedes/Archimedes_Finance
- Commit ID: fb95e183318f926fb11881cc8047f471ee89af90
- Smart contracts in scope:
    1. Auction.sol
    2. Coordinator.sol
        - coordinatorLvUSDTransferToExchanger()
        - acceptLeverageAmount()
        - resetAndBurnLeverage()
    3. Exchanger.sol
        - exchangerLvUSDTransferToCoordinator()
    4. ParamStore.sol
        - getArchToLevRatio()
        - changeCoordinatorLeverageBalance()

Out-of-scope:
- Third-party libraries and dependencies
- Economic attacks

2. Remediations Commit ID: auctionAuditFixes

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	2	0	7



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - LEVERAGE CAN BE ACQUIRED BEFORE THE AUCTION STARTS	Critical	SOLVED - 12/25/2022
HAL-02 - DUTCH AUCTION SYSTEM IS VULNERABLE TO FRONT-RUNNING ATTACKS	Medium	RISK ACCEPTED
HAL-03 - CLOSED POSITIONS ADD LEVERAGE TO THE COORDINATOR CONTRACT	Medium	SOLVED - 12/25/2022
HAL-04 - MISLEADING VARIABLE NAMES	Informational	FUTURE RELEASE
HAL-05 - SOLC 0.8.13 COMPILER VERSION CONTAINS MULTIPLE BUGS	Informational	FUTURE RELEASE
HAL-06 - INCOMPLETE NATSPEC DOCUMENTATION	Informational	FUTURE RELEASE
HAL-07 - CURRENTBALANCEONLVUSDCONTRACT VARIABLE HOLDS LVUSD BALANCE ON COORDINATOR CONTRACT	Informational	FUTURE RELEASE
HAL-08 - REDUNDANT INITIALIZATION OF UINT256 VARIABLES TO 0	Informational	FUTURE RELEASE
HAL-09 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational	FUTURE RELEASE
HAL-10 - OPEN TODOs	Informational	FUTURE RELEASE

FINDINGS & TECH DETAILS

3.1 (HAL-01) LEVERAGE CAN BE ACQUIRED BEFORE THE AUCTION STARTS - CRITICAL

Description:

As per Archimedes Documentation, every time a new Leverage Round is made available by Archimedes, an auction is set in place in order to determine the price of this leverage (by determining the ARCH purchase power). The price of the leverage decreases with time until it hits the minimum determined by auction parameters.

Once the auction is over (because there is no more leverage available or because the price hit the cap), the leverage price remains at the lowest point defined in the auction (no matter whether there is available leverage or not) until a new auction is started.

In order to start a new Leverage Round, the following is required:
- Sending or minting LvUSD to the Coordinator contract.
- Validating the Coordinator contract LvUSD balance by calling acceptLeverageAmount() function.
- Starting a new auction.

It has been detected that, since the minimum leverage price is not refreshed until the new auction is started, anyone could acquire the leverage added for the next Leverage Round at the lowest price if the balance of the Coordinator contract is validated before the new auction has been launched. This could be achieved by monitoring the Coordinator contract for acceptLeverageAmount() calls and opening a position before the new auction gets launched, front running startAuction() or startAuctionWithLenght() calls if needed.

Proof of Concept:

In this PoC, USER1 waits until an auction is over and the maximum archToLevRatio (meaning the lowest LvUSD price) is reached. After that, they monitor Coordinator contract looking for any acceptLeverageAmount() function calls from the owners indicating that more leverage has been added to the contract. After that, a new position is immediately opened based to the previous archToLevRatio before the new auction is created, taking the new leverage at the previous Leverage Round end price:

```
The previous auction is over. Only 1.53 LvUSD remains in the Coordinator contract, and the ArchToLev ratio is 10:1, the highest value defined in the auction:

Available leverage on Coordinator contract: 1.53423457

Current ARCH to Leverage ratio: 10.0

Is the current auction closed? : True
------------------------------------

Owners proceed to add 100 LvUSD for the next Leverage Round and accept it from Coordinator contract:

Sending 100 LvUSD to Coordinator --> contract_LvUSDToken.mint(1.53423457*10**18, {'from': OWNER})
Transaction sent: 0x3daedf95ad6824d21c4f3da7b7a57a03956d738c9502d2c4aac6a0c92d867caf
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 47
  LvUSDToken.mint confirmed   Block: 16236181   Gas used: 36940 (0.01%)

Accepting it from the validator contract --> contract_Coordinator.acceptLeverageAmount(101.53423457*10**18, {'from': OWNER})
Transaction sent: 0x41d78cf709f08a63ca4d7cc67b3761cd34cc5d6699ebfc861bf1b50918c9ba99
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 48
  Coordinator.acceptLeverageAmount confirmed   Block: 16236182   Gas used: 36579 (0.01%)


Now, owner starts a new auction --> contract_Auction.startAuctionWithLength(100, 1*10**18, 10*10**18, {'from': OWNER})

USER1 front-runs the previous call, opening a position with the recently added balance but using the max ArchToLev ratio from
the previous auction --> (100 OUSD, 1 Cycle, 10 ARCH provided) --> contract_LeverageEngine.createLeveragedPosition(100*10**18, 1, 10*10**18, {'from': USER1})

Transaction sent: 0x8baa70ca94a278cad7715bd7e7f1c2b06910623c54ce73f8e19c991a95a27f66
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 49
Transaction sent: 0xc758f9bd7d9c2be951eb157a8d34965f1d1b895c8b1d9b05350d053421cd60eb
  Gas price: 10.0 gwei   Gas limit: 600000000   Nonce: 2
  LeverageEngine.createLeveragedPosition confirmed   Block: 16236183   Gas used: 1205787 (0.20%)

  Auction.startAuctionWithLength confirmed   Block: 16236183   Gas used: 50173 (0.01%)

------------------------------------
Available leverage on Coordinator contract: 6.53423456999999
USER1 has successfully opened the position, so the Leverage has been taken:
USER1 PositionToken balance: 1

SINCE USER1 TOOK ~95 LvUSD WITH ONLY 10 ARCH, HE WAS ABLE TO TAKE LEVERAGE FROM NEW LEVERAGE ROUND AT THE MINIMUM PRICE SET FOR THE PREVIOUS ONE.
```

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

It is recommended to always ensure that a new auction has been started before confirming Coordinator's LvUSD balance, thus making the leverage publicly claimable by anyone.

This could be enforced on-chain by adding a require statement in the acceptLeverageAmount() function enforcing that isAuctionClosed() is false if the new leverageAmountToAccept value is greater than the value returned by getAvailableLeverage(), meaning that leverage is being added to the contract, not subtracted.

Remediation Plan:

SOLVED: The Archimedes Finance team solved the issue by adding a require statement in the acceptLeverageAmount() function to ensure that it only could be called when an auction is running.

Commit ID: 2cd099fa9ef8b6bddaf7a2fd7e687a35551236a5

3.2 (HAL-02) DUTCH AUCTION SYSTEM IS VULNERABLE TO FRONT-RUNNING ATTACKS - MEDIUM

Description:

The amount of LvUSD held in the Coordinator contract determines how much leverage users can take. The Auction contract implements a Dutch Auction mechanism to determine the amount of leverage per ARCH obtained when opening a position.

Each auction is defined by startPrice and endPrice (which will limit the minimum and maximum leverage amount obtained per ARCH) and endBlock, which determines the end of the auction once that block number is reached. Once an auction starts, the leverage price decreases every block (meaning a higher leverage per ARCH ratio), going from startPrice to endPrice in the time between block number at the time of starting the auction and the one defined in endBlock.

Since available leverage is scarce, leverage takers have to decide between taking the leverage in the early stages of the auction, maximizing the probability of winning the auction and taking the leverage at a higher price, or wait until the price decreases, which would return more leverage per ARCH but also increases the odds of someone else taking the available leverage, losing the auction.

It has been detected that any malicious leverage taker could cheat in the auction and always ensure the lowest leverage price possible. This could be done by monitoring the mempool once an auction has started for any bid and front-run it. By doing this, the malicious leverage taker would always win the auctions at the lowest price possible, defeating the purpose of the Dutch Auction mechanism.

Proof Of Concept:

In this proof of concept, two users, USER1 and USER2 compete against each other in an auction, going from 1e18 to 10e18 archToLevRatio in 100 blocks. Both users try to open a position with 100e18 OUSD as collateral, 1 cycle, and a maximum of 20e18 ARCH. The ‘Coordinator’ contract only holds 99e18 LvUSD.

1. Both USER1 and USER2 start with 100e18 ARCH and 100e18 OUSD respectively.
2. OUSD and ARCH approvals are signed by both users.
3. 99e18 LvUSD are sent to the Coordinator contract and the balance gets updated by calling acceptLeverageAmount().
4. An auction is created with 100 blocks duration.
5. USER1 waits until 95 blocks were mined, and tries to open a position with a 9.55 archToLevRatio. Since USER1 is the first one bidding in the auction, they should get the LvUSD.
6. USER2 detects USER1’s transaction in the mempool and front-runs it, cheating the auction system and winning it even though they placed the bid after USER1.

```
Sending 100 ARCH and 100 OUSD for USER1 and USER2...
Transaction sent: 0x7b63dd8cc4f25b72d68b4c65474a50cb04d9453e4d7d0631e6479da773fa7495
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  ArchToken.transfer confirmed   Block: 16228562   Gas used: 52913 (0.01%)

Transaction sent: 0x2ab5c5c3673f9c039808daff7531feb7af7c4cfa3b1c721c4393b0726972b4e4
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ArchToken.transfer confirmed   Block: 16228563   Gas used: 52913 (0.01%)

Transaction sent: 0x2ebd218ce15a2ebb986dc315a15cc9e0f5b096e8a70a6e5c0b5a0fab4c932629
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 2
  InitializeGovernedUpgradeabilityProxy.transfer confirmed   Block: 16228564   Gas used: 78584 (0.01%)

Transaction sent: 0xe3a094574a0e139605ca3d40fc1a3fbf15e4fc1b23579c5faf81cf2852504261
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 3
  InitializeGovernedUpgradeabilityProxy.transfer confirmed   Block: 16228565   Gas used: 78584 (0.01%)
Setting needed approvals...
Transaction sent: 0xb70964e240583cc8a7db6eff7dd30039afe3ba5e9138c03b4067e7769af83deb
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  InitializeGovernedUpgradeabilityProxy.approve confirmed   Block: 16228566   Gas used: 46010 (0.01%)

Transaction sent: 0x17c456a5cc37e6ac2d618281dc92d8f5e67090355c90f24a865598272c583999
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ArchToken.approve confirmed   Block: 16228567   Gas used: 44225 (0.01%)

Transaction sent: 0x2d03d550aeca17a216ccc397c1faf0c7457323d904f0211a82906d27929b0980
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  InitializeGovernedUpgradeabilityProxy.approve confirmed   Block: 16228568   Gas used: 46010 (0.01%)

Transaction sent: 0x4c3c3ab88e790feb3c3725a9e0c1c9e964b473ecafb87468212e4ff5391a9a92
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ArchToken.approve confirmed   Block: 16228569   Gas used: 44225 (0.01%)
```

```
Minting 99 LvUSD to Coordinator Contract --> contract_LvUSDToken.mint(99*10**18, {'from': OWNER})
Transaction sent: 0x2feee8eeb877d306300be442660cfe8fb90435c189cfa7b2b6b35ed9e33313dd
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 43
  LvUSDToken.setMintDestination confirmed   Block: 16228570   Gas used: 28544 (0.00%)

Transaction sent: 0x5a648f0560b13727010d9bc17465437434811de2f86e14437e23740d7c23df70
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 44
  LvUSDToken.mint confirmed   Block: 16228571   Gas used: 51940 (0.01%)

Updating leverage amount on Coordinator contract --> contract_Coordinator.acceptLeverageAmount(99*10**18, {'from': OWNER})
Transaction sent: 0xc2495402b5f4ed066f5af1dab3c866e47153a14b5bcbd98587ac96aa21d395f2
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 45
  Coordinator.acceptLeverageAmount confirmed   Block: 16228573   Gas used: 51555 (0.01%)

LvUSD balance on Coordinator contract: 99.0
Available leverage on Coordinator contract: 99.0
------------------------------------

USER1 OUSD Balance: 100.0
USER1 ARCH Balance: 100.0
------------------------------------

USER2 OUSD Balance: 100.0
USER2 ARCH Balance: 100.0
------------------------------------
Creating Auction with 1 to 10 ARCH to Leverage ratio --> contract_Auction.startAuctionWithLength(100, 1*10**18, 10*10**18, {'from': OWNER})
Transaction sent: 0x1143a190341b70e1da43bd61a290239a6fe5e68c3970a4c717eec9cb6679fa65
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 46
  Auction.startAuctionWithLength confirmed   Block: 16228574   Gas used: 121963 (0.02%)

------------------------------------

Waiting 95 blocks...
------------------------------------

Current ARCH to Leverage ratio: 9.55
------------------------------------

USER1 opens a new position (100 OUSD, 1 Cycle, 10 ARCH provided) --> contract_LeverageEngine.createLeveragedPosition(100*10**18, 1, 20*10**18, {'from': USER1})

USER2 sees this transaction in the mempool and frontruns it, setting a higher gas price --> contract_LeverageEngine.createLeveragedPosition(100*10**18, 1, 20*10**18, {'from': USER2})

Transaction sent: 0x7951d7ce0913d44ba2abf0c0fe4d94a2972c508a9d46cea69c14b797a3f29477
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 2
Transaction sent: 0xb7a97e0ad88a1a74ea3565bcdc4902ae820bcd6b6e680b59e5f2326f1b223528
  Gas price: 10.0 gwei   Gas limit: 600000000   Nonce: 2
  LeverageEngine.createLeveragedPosition confirmed   Block: 16228670   Gas used: 1214785 (0.20%)

  LeverageEngine.createLeveragedPosition confirmed (Not enough available leverage)   Block: 16228670   Gas used: 69335 (0.01%)

------------------------------------
Available leverage on Coordinator contract: 4.0
Since the available leverage has decreased, one of the position has been successfully opened:
USER1 PositionToken balance: 0
USER2 PositionToken balance: 1

THE CREATE POSITION CALL HAS BEEN SUCCESSFULLY FRONTRUN AND USER2 SUCCESSFULLY CHEATED ON THE AUCTION.
```

Risk Level:

Likelihood - 3

Impact - 4

Recommendation:

The best remediation against front-running attacks is to remove the benefit for the attacker of front-running transactions from the application by eliminating the importance of transaction ordering or timing. However, if this is not possible, additional mitigations could be implemented:

- A gas price upper limit would prevent users from specifying a higher gas price to front-run legitimate bids. However, this solution would require permanent supervision according to the gas price fluctuations if the threshold is tight enough and could even prevent legitimate bids from being processed.

- Off-chain ordering mechanisms allow application owners to split the transactions into two separate phases: Ordering (that is performed off-chain) and settlement (performed on-chain). On the other hand, this makes the application flow less transparent for users.
- Commit and reveal strategies such as Submarine Sends could conceal not only the content of the transactions, but also their mere existence. However, this approach would add additional complexity to the application, increasing transaction time and cost.
- Transaction sequencing mechanisms could prevent front-running attacks by generating off-chain signed messages containing a unique counter value that would be included in contract calls, reversing calls containing a counter value that does not match with the one from the contract. msg.sender value should also have to be included to prevent signature stealing.

Remediation Plan:

RISK ACCEPTED: The Archimedes Finance team accepted the risk of this finding, considering this an edge case that would have a negligible impact on the users. Also, Archimedes claims that early checks will be implemented in the future to minimize the gas cost for reverted transactions affected by this issue.

3.3 (HAL-03) CLOSED POSITIONS ADD LEVERAGE TO THE COORDINATOR CONTRACT - MEDIUM

Description:

Archimedes Documentation describes that Archimedes makes leverage available to users through Leverage Rounds. In these Leverage Rounds, a certain amount of LvUSD is deposited and validated in the Coordinator contract. With each Leverage Round, an auction is started, determining the price of that leverage (price decreases over time).

Once a user claims that leverage by opening a position, the associated LvUSD is transferred to the 3CRV/LvUSD pool in exchange for 3CRV tokens (that are transferred to the 3CRV/OUSD pool in exchange for more OUSD). To complete this, the _coordinatorLvUSDTransferToExchanger() function is used to withdraw LvUSD from Coordinator contract, ensuring that _coordinatorLeverageBalance (validated and usable leverage in Coordinator contract) gets properly updated.

However, it has been detected that when any user closes a position (hence exchanging the OUSD leverage for LvUSD to pay for the borrowed leverage), the obtained LvUSD is transferred back to the Coordinator contract with the _exhangerLvUSDTransferToCoordinator() function, which also updates the _coordinatorLeverageBalance value, instantly making more leverage available without having to wait for the next Leverage Round.

Since Archimedes cannot usually predict when users close their positions (unless positions automatically unwind after 370 days, according to the documentation), this leads to adding leverage at uncontrolled prices, depending on the state of the auction at the time (if any).

Code Location:

Listing 1: Exchanger.sol (Line 115)

```
109     function _exhangerLvUSDTransferToCoordinator(uint256 amount)
    ↳ internal {
110         /// Is it possible to exploit via transferring lvUSD to
    ↳ exchanger which then go back to coordinator?
111         uint256 currentExchangerLvUSDBalance = _lvUSD.balanceOf(
    ↳ address(this));
112         uint256 currentCoordinatorLeverageBalance = _paramStore.
    ↳ getCoordinatorLeverageBalance();
113         require(currentExchangerLvUSDBalance >= amount, "insuf
    ↳ lvUSD to trnsf to Exhanger");
114
115         _paramStore.changeCoordinatorLeverageBalance(
    ↳ currentCoordinatorLeverageBalance + amount);
116         _lvUSD.safeTransfer(_addressCoordinator, amount);
117     }
```

Proof of Concept:

In this simple PoC, a new Leverage Round is created, with 100 LvUSD available. USER1 opens a position to check that the claimed leverage is properly subtracted from _coordinatorLeverageBalance and then the position is closed. The returned LvUSD will be instantly available without needing to be validated by Archimedes:

```
____________________________________
LvUSD balance on Coordinator contract: 100.0
Available leverage on Coordinator contract: 100.0
____________________________________

Starting an auction to set Leverage price --> contract_Auction.startAuctionWithLength(1, 10*10**18, 10.0000000001*10**18, {'from': OWNER})
Transaction sent: 0xbe9d05c1fa6cf6496498ae2b3f9430610e5f4f2526dc9dbfc60c54f15e98f835
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 46
  Auction.startAuctionWithLength confirmed   Block: 16242398   Gas used: 121975 (0.02%)

Opening a position for USER1, taking leverage --> openTx = contract_LeverageEngine.createLeveragedPosition(50*10**18, 2, 20*10**18, {'from': USER1})
Transaction sent: 0x56874b8629cfb7ef65e6a537422db097bb0139cc8214802ba66b4d55444c68e1
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 2
  LeverageEngine.createLeveragedPosition confirmed   Block: 16242399   Gas used: 1247073 (0.21%)

____________________________________
LvUSD balance on Coordinator contract: 7.375
Available leverage on Coordinator contract: 7.375
____________________________________

Closing the position --> closeTx = contract_LeverageEngine.unwindLeveragedPosition(0, {'from': USER1})
Transaction sent: 0x8e559461a2ad013beeac3df6d2a5a56c75a20e1088094d566c8f26ca652f43f2
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 3
  LeverageEngine.unwindLeveragedPosition confirmed   Block: 16242400   Gas used: 765538 (0.13%)

____________________________________
LvUSD balance on Coordinator contract: 100.99180884098848
Available leverage on Coordinator contract: 100.99180884098848
____________________________________

New leverage has been added without needing to wait for a new Leverage Round.
```

Risk Level:

Likelihood - 5
Impact - 2

Recommendation:

If this is not the intended behavior and Archimedes wants to fully control when new leverage becomes available, LvUSD could be only transferred to Coordinator contract without being validated by calling changeCoordinatorLeverageBalance().

Remediation Plan:

SOLVED: The Archimedes Finance team solved the issue by burning the LvUSD obtained when a position is closed.

Commit ID: 7c8c4b6d137c05dbca5581a46ea792677eded2a1

3.4 (HAL-04) MISLEADING VARIABLE NAMES - INFORMATIONAL

Description:

As described in HAL-01, the amount of leverage obtained per ARCH is calculated in a Dutch Auction. In such auction, the leverage purchase power of ARCH gets gradually increased over time until the auction ends or the leverage is sold out, as per Archimedes Documentation.

However, it has been detected that variable names used in the ARCH purchase power calculation function, _calcCurrentPriceOpenAuction(), make reference to prices instead of purchasing power or ARCH to leverage ratio. Since those are inversely proportional variables, it might lead to confusion and misuse, while also decreasing code readability.

Code Location:

Listing 2: Auction.sol

```
    function _calcCurrentPriceOpenAuction() internal view returns (uint256 auctionPrice) {
        /// y = ax + b
        /// => y = current auction price.
        /// linear graph show price in Y and price is going down over time so we'll need y = -ax + b
        /// might be easier to think about this as y = b - ax
        /// time is the X axis here so when we start auction t=0, when we end t=delt(startBlock, endBlock)
        /// we want to get time that is between 0 and 1 so we'll do
        /// so this means t_current = (currentBlock - startBlock)/ delta(startBlock, endBlock)
        /// b = startPrice. b has to equal startPrice since t=0 at  that point
        /// a = (startingPrice - endPrice)
        // currentPrice =  b - ax = startPrice - (startingPrice - endPrice) * t(0...1 only)
        uint256 deltaInPrices = _endPrice - _startPrice;
```

```
        uint256 deltaInPriceMulCurrentTime = (deltaInPrices * (block.number - _startBlock)) / (_endBlock - _startBlock);
        uint256 maxPriceForAuction = _startPrice;
        uint256 currentPrice = maxPriceForAuction + deltaInPriceMulCurrentTime;

        return currentPrice;
    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use appropriate variable names that accurately describe their intended function and make sense within the application logic flow.

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

3.5 (HAL-05) SOLC 0.8.13 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

The scoped contracts have configured the fixed pragma set to 0.8.13. The latest solidity compiler version, 0.8.17, fixed important bugs in the compiler along with new efficiency optimizations.

The official Solidity recommendations are: when deploying contracts, the latest released version of Solidity should be used. Apart from exceptional cases, only the latest version receives security fixes.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the latest Solidity compiler version as possible.

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

3.6 (HAL-06) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

Natspec documentation are useful for internal developers that need to work on the project, external developers that need to integrate with the project, auditors that have to review it but also for end users given that many chain explorers have officially integrated the support for it directly on their site.

It has been detected that no contract has a complete **natspec** documentation, and functions are little to no documented.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding the missing **natspec** documentation.

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

3.7 (HAL-07) CURRENTBALANCEONLVUSDCONTRACT VARIABLE HOLDS LVUSD BALANCE ON COORDINATOR CONTRACT - INFORMATIONAL

Description:

Two different variables are defined within _coordinatorLvUSDTransferToExchanger () function:
- currentCoordinatorLeverageBalance stores the amount of usable leverage stored in Coordinator contract.
- currentBalanceOnLvUSDContract stores the LvUSD balance of Coordinator contract.

However, currentBalanceOnLvUSDContract variable name is misleading and decreases code readability.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider renaming currentBalanceOnLvUSDContract to currentCoordinatorLvUSDBalance or a similar name that accurately depicts the intended purpose of the variable.

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

3.8 (HAL-08) REDUNDANT INITIALIZATION OF UINT256 VARIABLES TO 0 - INFORMATIONAL

Description:

As i is an uint256, it is already initialized to 0. uint256 i = 0 reassigns the 0 to i which wastes gas.

Code Location:

ParameterStore.sol
- Line 210:

```
for (uint256 i = 0; i < numberOfCycles; ++i){
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not initialize uint variables to 0 to save some gas. For example, use instead:

```
for (uint256 i; i < numberOfCycles; ++i){
```

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

3.9 (HAL-09) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL

Description:

Failed operations in this contract are reverted with an accompanying message containing a hardcoded string.

In the EVM, emitting a hardcoded string in an error message costs ~50 more gas than emitting a custom error. Additionally, hardcoded strings increase the gas required to deploy the contract.

Code Location:

- Auction.sol
  Line 46, Line 70, Line 116, Line 117, Line 118, Line 165, Line 172.
- ParameterStore.sol
  Line 82, Line 83, Line 91.
- Exchanger.sol
  Line 113.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Custom errors are available from Solidity version 0.8.4 up. Consider replacing all revert strings with custom errors. Consider also reviewing additional contracts and functions beyond the scope of this report for additional occurrences of this finding.

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

3.10 (HAL-10) OPEN TODOs - INFORMATIONAL

Description:

Open To-dos can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

Listing 3: Coordinator.sol

```
            /// TODO: add slippage protection
```

Listing 4: ParameterStore.sol

```
            /// TODO : Should this return a revert? Most likely but other changes are needed as well. This can be misleading
```

Listing 5: Exchanger.sol

```
            /// TODO Approval & Allownace should NOT BE MAX VALUES for pools
```

Listing 6: Exchanger.sol

```
            /// TODO : Should this return a revert? Most likely but other changes are needed as well. This can be misleading
```

Listing 7: Exchanger.sol

```
        // TODO allow user to override this protection
        // TODO auto balance if pool is bent
```

Listing 8: Exchanger.sol

```
        // TODO allow user to override this protection
        // TODO auto balance if pool is bent
```

Listing 9: Exchanger.sol

```
        // TODO allow user to override this protection
        // TODO auto balance if pool is bent
```

Listing 10: Exchanger.sol

```
        // TODO allow user to override this protection
        // TODO auto balance if pool is bent
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving the To-dos before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

PENDING: The Archimedes Finance team will solve this issue in future releases.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. 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.

Results:

Auction.sol

```
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204) uses delegatecall to a input-controlled function id
	- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

Auction (contracts/Auction.sol#10-175) is an upgradeable contract that does not protect its initiliaze functions: Auction.initialize() (contracts/Auction.sol#142-151). Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(address) (node_modules/@openzeppelin/contracts-upgr
adeable/proxy/utils/UUPSUpgradeable.sol#72-75)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#85-88)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-con
tract

Auction.getCurrentBiddingPrice() (contracts/Auction.sol#58-74) uses a dangerous strict equality:
	- biddingPrice == 0 (contracts/Auction.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by IERC1822ProxiableUpgradeable(newImplementation).proxiableUUID() (node_modules/@openzeppe
lin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98)' in ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/c
ontracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) potentially used before declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upgrade: unsupported proxiableUUID) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeU
pgradeable.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#52-57) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#54-56)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#62-67) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#72-77) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#82-87) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#84-86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Auction._startAuction(uint256,uint256,uint256) (contracts/Auction.sol#41-51) compares to a boolean constant:
	-require(bool,string)(isAuctionClosed() == true,err:auction currently running) (contracts/Auction.sol#46)
Auction.isAuctionClosed() (contracts/Auction.sol#103-109) compares to a boolean constant:
	-_isAuctionClosed == true || _endBlock < block.number (contracts/Auction.sol#104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity are used:
	- Version used: ['0.8.13', '^0.8.0', '^0.8.1', '^0.8.2']
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4)
	- ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4)
	- ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4)
	- ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
	- 0.8.13 (contracts/AccessController.sol#2)
	- 0.8.13 (contracts/Auction.sol#2)
	- 0.8.13 (contracts/interfaces/IAuction.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#220-224) is never used and should be removed
AccessControlUpgradeable._setupRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#211-213) is never used and should be removed
AccessController._setAndRevokeAnyRole(bytes32,address,address) (contracts/AccessController.sol#103-110) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#114-120) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65) is never used and should be removed
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is never used and should be removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#139-142) is never used and should be removed
ERC1967UpgradeUpgradeable._getAdmin() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#122-124) is never used and should be removed
ERC1967UpgradeUpgradeable._getBeacon() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#158-160) is never used and should be removed
ERC1967UpgradeUpgradeable._setAdmin(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#129-132) is never used and should be removed
ERC1967UpgradeUpgradeable._setBeacon(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#165-172) is never used and should be removed
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#180-190) is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#131-137) is never used and should be removed
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#72-77) is never used and should be removed
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#82-87) is never used and should be removed
StringsUpgradeable.toHexString(address) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#72-74) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#41-52) is never used and should be removed
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#16-36) is never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4) allows old versions
Pragma version0.8.13 (contracts/AccessController.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (contracts/Auction.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (contracts/interfaces/IAuction.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Low level call in ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is not in mixedCase
Variable AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#259) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#211) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#23-24) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is not in mixedCase
Variable UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#107) is not in mixedCase
Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is not in mixedCase
Variable ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#41) is not in mixedCase
Variable AccessController.__gap (contracts/AccessController.sol#27) is not in mixedCase
Variable Auction._addressAuctioneer (contracts/Auction.sol#13) is not in mixedCase
Variable Auction._currentAuctionId (contracts/Auction.sol#16) is not in mixedCase
Variable Auction._startBlock (contracts/Auction.sol#17) is not in mixedCase
Variable Auction._endBlock (contracts/Auction.sol#18) is not in mixedCase
Variable Auction._startPrice (contracts/Auction.sol#19) is not in mixedCase
Variable Auction._endPrice (contracts/Auction.sol#20) is not in mixedCase
Variable Auction._isAuctionClosed (contracts/Auction.sol#22) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#107) is never used in Auction (contracts/Auction.sol#10-175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

grantRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#150-152)
revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#165-167)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#185-189)
        - AccessController.renounceRole(bytes32,address) (contracts/AccessController.sol#70-77)
setAdmin(address) should be declared external:
        - AccessController.setAdmin(address) (contracts/AccessController.sol#49-55)
getAddressExecutive() should be declared external:
        - AccessController.getAddressExecutive() (contracts/AccessController.sol#112-114)
initialize() should be declared external:
        - Auction.initialize() (contracts/Auction.sol#142-151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- Issues found by Slither are either already reported or false positives.

AUTOMATED TESTING

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

Auction.sol

Report for contracts/Auction.sol
https://dashboard.mythx.io/#/console/analyses/9bcf29d3-513a-4016-9d74-17dead0b7002

Line	SWC Title	Severity	Short Description
29	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
94	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
104	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
116	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
127	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.

- No major issues were found by MythX.

THANK YOU FOR CHOOSING

HALBORN