QuillAudits

Audit Report
December, 2023

For

Carpe Diem
Pension

Table of Content

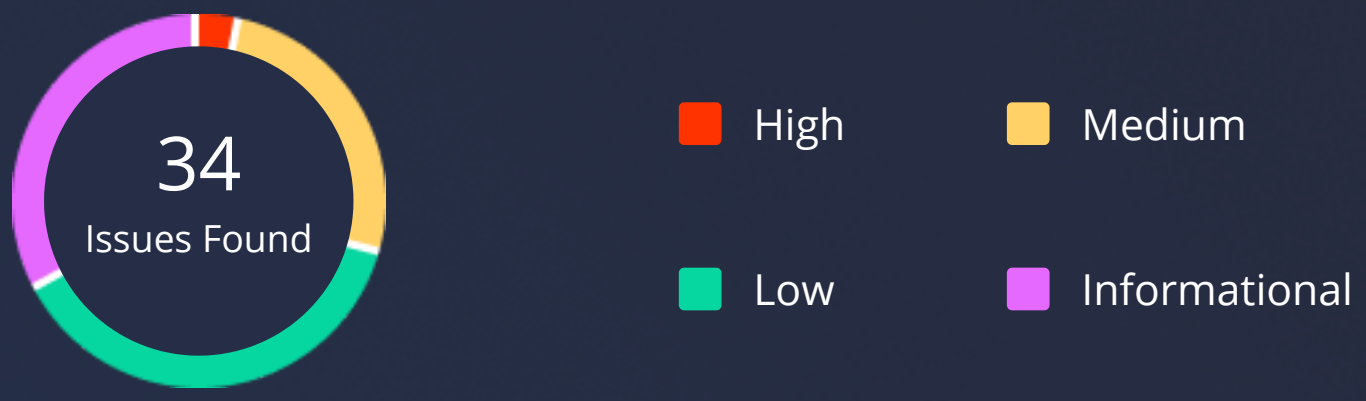
Table of Content

Table of Content

Executive Summary

Project Name	Carpe Diem
Project URL	*https://carpediempension.com/*
Overview	Carpe Diem Pension (CDP) protocol contains Pension, Auction and CDPtoken contract. The CDP token can be deposited in a Pension contract, and rewards can be earned on it. Users can deposit in auctions with native blockchain tokens and would be able to collect shares, which then can be used to earn CDP tokens.
Timeline	30th October 2023 - 28th November 2023
Updated Code Received	8th December 2023
Second Review	11th December 2023 - 18th December 2023
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the Carpe Diem codebase for quality, security, and correctness.
Source Code	*https://github.com/CarpeDiemCDP/Pension/tree/main/contracts*
Branch	main
Commit	5ec937f689d82d45019767634b84d6b497eab768
Fixed In	cad3018fc34ac433643cef4c2fb914b376a25361
Blockchain	To be deployed on PulseChain

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	5	5
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	8	8	6

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high-severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. User would be able to claim/compound very big amount without depositing and waiting for days/years

Path

Pension.sol

Function

depositCDP()

Description

While setting **_referral** in **depositCDP()** because snapshot is not getting set for **_referral** address, the **_referral** would be able to create the scenario where while calculating pending in **claimCDP** or **compoundCDP**, in this formula **pending += (rewardPerShare - snapshot) * userShares + storedReward;** the snapshot would be 0 because that was never set for referral, **(rewardPerShare - snapshot)** would be **rewardPerShare** value which will get multiplied with **userShares** (referral's share in this case) so that **_referral** would be able to claim rewards for all days even after getting referral very recently.

Here, two scenarios can happen:

1. Users can claim a very big reward without waiting for a long time (only because the snapshot was not set while giving a referral).
2. Users would be able to compound and create a pending amount from which 1.15 (for every 1) would be added to the user share, they would be able to create a very big user share without waiting.

Recommendation

Consider recording a snapshot for a referral address after L123. e.g. **userInfo[ref].snapshot=rewardPerShare;** and also calculate reward to store for referral address e.g.

```
if (referralShares != 0 && referralSnapshot < rewardPerShare) {
userRef.storedReward += (rewardPerShare - referralSnapshot) * referralShares;
}
```

It needs to be added for a referral because if the referral has any previous rewards, it wont get stored as the snapshot will again get recorded for that referral.
If remains unfixed, then any user can intentionally or unintentionally erase the accumulated rewards for any user because the snapshot for that referral will get reset.

Status

Resolved

Medium Severity Issues

2. Fix the syntax error on L110 and L210 (212 on updated contract version)

Path

Pension.sol#L110,#L210

Function

_mintDailyCDPandShares()

Description

Add parenthesis on lines 110 and 210 (can be 212 on updated contract version) if condition (in the new commit on patch-2 https://github.com/CarpeDiemCDP/Pension/blob/ab287c6a0987518de00bab0d1b7e4e32e76afc50/contracts/Pension.sol#L212 its still unfixed).

Recommendation

Fix the error by adding or removing parentheses brackets.

Status

Resolved

3. Precision loss + unscaled value while calculating rewardPerShare in mintCDP()

Path

Pension.sol#L233, Auction.sol#L274, L275

Function

mintCDP()

Description

While calculating rewardPerShare when _amount would be smaller than (totalShares - auctionShares - activeAuctionShares), the result that will get added to the rewardPerShare would be 0. It will affect the functionality to find the pending amount because the function(s) check if "(snapshot < rewardPerShare)" is true, then it calculates pending, which then gets sent to msg.sender. But because of precision loss, the rewardPerShare will be 0. E.g. _amount=52479574642711924983241 and (totalShares - auctionShares - activeAuctionShares)=60346100000000000000001, so the answer is 0.8694752658924353 37, but because solidity doesn’t support decimals, it will be only 0.

Recommendation

- Here, we can multiply the x (52479574642711924983241 in this case) with 1e18 and then will divide that value with the y (60346100000000000000001 in this case) so the result in this specific example would be 8694752658924353 37 (which is 0.8694752658924353 37 in ETH form). This is similar to the way that OZ FixedPointMathLib lib's divWadDown uses to prevent precision loss.
- Now, If initially when there would be no deposits, the totalShares would be 1, But as the numerator (_amount) is getting multiplied with 1e18, we need to have the denominator as 1e18 to **normalize/scale down** the output (when there won't be any deposit)

  E.g _amount is 52479574642711924983241.

  So (52479574642711924983241*1e18)/1 = 52479574642711924983241e18 (i.e 524795746427119249832410000000000000000), which is incorrect output.

  So that's why totalShares needs to be initialized as 1e18 and not 1 on L15

  The code in mintCDP() would look like this:

```
function mintCDP(
        uint256 _amount,
        uint256 _day
    ) external {
        require(msg.sender == AuctionContractAddress);
        CDP.mint(address(this), _amount);
        if(totalShares==1e18){
            rewardPerShare += ((_amount*1e18 )/ (totalShares - auctionShares -
activeAuctionShares));
        } else {
            uint denom=((totalShares-1e18) - auctionShares - activeAuctionShares);
            if (denom==0){
                rewardPerShare += _amount;
            }else {
                rewardPerShare += (_amount*1e18 )/ denom;
            }
        }
        dayInfoMap[_day].CDPRewards = _amount;
        dayInfoMap[_day].totalShares = totalShares;
```

```
    NoUsersPerDay[_day] = NoUsers;
  }
```

So here, if else statements need to be used because when totalShares would be equal to 1e18 that means no one has deposited and to scale down **(_amount*1e18)** it needs to be divided with **totalShares** (1e18).

When totalShares won't be equal to 1e18, that means it's greater and the shares are there, which means users have deposited them, in that case, it checks if the denominator is 0 and then directly adds **_amount** to **rewardPerShare**. The denominator can be 0 even when totalShares would be greater than 1e18, i.e. in the case when totalShares -1e18 would be equal to **auctionShares + activeAuctionShares**.

Else, rewardPerShare can be calculated with that denom.

- Because we are initializing totalShares to 1e18 in the Auction contract in todayMintedCDP() while finding the return value (((totalSupply + historicSupply) * 10000) / 103563452) we can't take the totalShares as the part of historicSupply because the value of totalShares is 1e18 (and not very small i.e 1). So following changes needs to be made:

```
if(totalShares==1e18){
        return (((totalSupply ) * 10000) / 103563452);
} else {
        uint256 historicSupply = ((totalShares - 1e18) * 10) / 13;
return (((totalSupply + historicSupply) * 10000) / 103563452);
 }
```

So Here historicSupply only gets used when totalShares greater than 1e18 but while using 1e18 needs to be subtracted from it so that it should not be the part of historicSupply.

Auditor's Comment

It is currently fixed by adding denom != 0 condition, if denom is not equal to 0 then only it will mint CDP tokens and increase rewardPerShare. So even though Auction.doDailyUpdate() will happen and Pension.mintCDP() will be executed, the rewardPerShare won't be incremented for that day if denom is 0. and CDP _amount won't be minted for that day.

Status

Resolved

4. Multiplication can give incorrect answers because of unscaled values

Path

Pension.sol#L113, L151, L176, L213, L265 and L288

Function

mintCDP()

Description

On L113, L151, L176, L213, L265 and L288 while multiplying, we need to use similar functionality like mulWadDown function which will do (a*b)/1e18 to normalise the output.

E.g In compoundCDP() on L176 for pending calculation **pending += (rewardPerShare - snapshot) * userShares + storedReward** in **(rewardPerShare - snapshot) * userShares** in calculation the decimals get scaled up e.g if **(rewardPerShare - snapshot)** is **5e18** and **userShares** is **2e18** then the output **should be** 10e18 (10000000000000000000), but the value here **would be** 100000000000000000000e18 (i.e 10000000000000000000000000000000000000). So, this value needs to be scaled down.

Recommendation

To scale down, the value needs to be divided by 1e18, which would look like this (((rewardPerShare - snapshot) * userShares)/1e18).

The fix should be added to the above-mentioned line numbers where this calculation is getting used. Note: Don’t forget to add storedReward after it wherever it is present in the current calculation.

Status

Resolved

5. Protocol allocate shares for all days, even when there are no deposits

Path

Auction.sol#L248-L250

Function

_mintDailyCDPandShares()

Description

The Protocol assumes that shares are distributed to users. But in the case that no one has participated in the day's auction, the appointed shares are still burnt (and not

compensated/minted for users). And, therefore, would never be withdrawn.
This could also lead to more consequences, including accounting errors and unexpected behaviours.

Proof of Concept

doDailyUpdate() can be called at the END of the day, which implies at the end of an auction, and takes 5% of the total auctionshares and allocates it as shares to be distributed for that concluded auction.

```
// shares that belong to auction are burned as they are distributed in Pension to
users
        uint256 nextDayShares = PensionContractInterface(PensionContractAddress)
            .BurnSharesfromAuction();
        shares[_day] = nextDayShares; // this is the amount of shares that are for
sale on _day
```

If no one participated that day, it would still allocate 5% of the total auctionshares for that day, which would never be withdrawn.

Recommendation

Check if PLSauctionDeposits[_day] != 0

```
if (PLSauctionDeposits[_day] != 0) {
        // shares that belong to auction are burned as they are distributed in Pension
to users
        uint256 nextDayShares = PensionContractInterface(PensionContractAddress)
            .BurnSharesfromAuction();
        shares[_day] = nextDayShares; // this is the amount of shares that are for
sale on _day
}
```

Status

Resolved

6. Users would always have their user.referral remain address(0), due to a logic error

Path

Pension.sol#L96

Function

depositCDP()

Description

Users would always have their user.referral remain address(0), due to wrong if-else logic; since it will never reach case 3.

Proof of Concept

When entered _referral is != address(0) and having user.referral as zero initially since that's the default initial value, then the second else if() condition will be true, so L96 will execute, and last else would not get executed even while having user.referral as zero initially.

```
} else if (
    _referral == user.referral || user.referral == address(0) // @audit
) {
    // Set the ref to the provided referral
    ref = _referral;
} else {
```

Recommendation

Consider each case and ensure none is inconsistent with the intended logic.

Status

Resolved

7. Accounting errors for the protocol due to totalBurned not being correctly tracked

Path

CDPToken.sol#L30

Function

burn(), burnFrom()

Description

totalBurned is used to represent how many CDP have been burned, this variable read off and used on the frontend and be external applications. This is tracked on the burn()

function However failed to override and account on the public burnFrom function as well which is inherited from the openzeppelin ERC20Burnable. This causes incorrect accounting on the front end and external applications leading to unexpected behaviours.

Proof of Concept

The burn() function which is used by the protocol, updates the totalBurned variable before burning. However, protocol also specifies that this function can be called directly
`People or systems can use the burn function to directly burn CDP, skipping the creation of shares.`
This can be done using the burn and burnFrom function since openzeppelin ERC20Burnable is inherited

```
function burnFrom(address account, uint256 value) public virtual {
        _spendAllowance(account, _msgSender(), value);
        _burn(account, value);
    }
```

but only accounted for the burn() function.

From the Deployment Documentation

```
People or systems can use the burn function to directly burn CDP, skipping the
creation of shares. This function is mainly for systems, as those are often not able
to claim rewards or don't have intention to gain rewards at the first place. The
function can be used by communicating directly with the smart contract or by
using a block explorer that displays all the functions. Carpe Diem doesn't
implement this function in its main user interface, as it might lead to confusion.
```

Recommendation

Override the burnFrom() function and account for it as well.

Status

Resolved

8. Add a check for the minimum amount that should be deposited

Path

Pension.sol

Function

depositCDP()

Description

depositCDP() on L80 checks **if (_amount > 0)** then only it executes the code block. But it can happen that the user will put a really small amount in wei but greater than 0, e.g 2. So, in this case on the L118 user. shares for that user will get updated successfully but when it will try to find **auctionShares** by multiplying by 2 and dividing by 10 then (2*2)/10 would be 0 (in SOL), which means **auctionShares** would be 0.

Recommendation

We recommend adding a limit instead of only checking **_amount** to be greater than 0 so that all things can be calculated successfully without rounding the answer to zero because solidity doesn't support decimals.

Status

Resolved

9. Automate doDailyUpdate() call for the 20hr interval so that it won't skip any day

Path

Auction.sol

Function

doDailyUpdate()

Description

Skipping doDailyUpdate() can create some unintended scenarios where some state changes won't be reflected before.

e.g Users won't be able to collect auction shares as collectAuctionShares() checks for **require(targetDay < currentDay, "cant collect tokens for current active day");** condition to be true. so if **doDailyUpdate()** won't execute for some days then for those days user won't be able to collect shares even after entering the auction by paying the PLS amount.

Pension contract functionality can also have some unexpected results e.g because **rewardPerShare** won't get updated for the skipped day, in that interval user can use some functionalities but for some functionalities like claimCDP() user won't be able to claim the rewards when **rewardPerShare** and **snapshot** would be same because of failure to call **doDailyUpdate()** on time (every 20 hrs).

Recommendation

Automate the doDailyUpdate() call for the 20 hr interval using solution like Chainlink Automation **https://docs.chain.link/chainlink-automation** (**https://docs.chain.link/chainlink-automation/overview/getting-started**)

Carpe Diem Team's Comment

As CDP would be minted for skipped days in next doDailyUpdate() call, users would be able to collect user shares for entered PLS and would be able to claim rewards for skipped daily 'cdp minting' later once doDailyUpdate() executes. Additionally the team will contact oracle service to check the support for Pulsechain.

Auditor's Comment

Even though mintCDP() will execute for skipped days, it should be noted that while calculating amounts such as rewardPerShare, denom will get calculated according to current state e.g totalShares at that moment can be different than that was for any specific skipped day. Apart from that if a user performs actions like Destroyshares() in the interval where day are getting skipped, then the pending amount for that user will get calculated according to outdated rewardPerShare and snapshot.

Status

Acknowledged

10. getStatsLoops adds static 1 instead of i

Path

Auction.sol

Function

getStatsLoops()

Description

On L319 in **sharesDay[i] = shares[_day + 1];** assignment, while finding shares for the **_day , 1** is getting added instead of i which will return the same amount each time for every iteration.

Recommendation

Consider verifying the logic and add **i** instead of **1**.

Status

Resolved

Low Severity Issues

11. User can trick the referral to extract more CDP from the protocol

Path

Pension.sol#L79

Function

depositCDP()

Description

In the depositCDP() function, specifically on line 79, there is a check in place to ensure that msg.sender cannot be the same as the _referral the user provided. It's important to highlight that despite this validation, a user has the ability to input ANY other address as the _referral. This allows the user to effectively claim the CDP for themselves, bypassing the intended restriction.

Recommendation

Confirm that the team is aware about this scenario.

Carpe Diem Team's Comment

We are aware that users could refer themselves through the use of multiple addresses.

Status

Acknowledged

12. Unutilized totalRefShares and Missing Decrement on Share Destruction

Path

Pension.sol#L292-L299

Function

Destroyshares()

Description

The variable totalRefShares appears to be unused but more importantly there is an issue where it does not get decremented when a referred user shares is destroyed. This oversight raises concerns regarding the proper tracking of overall shares, potentially impacting the accuracy of calculations or leading to unintended consequences in the system due to this inconsistency.

Proof of Concept

The Destroyshares() function deletes a user's shares if he hasn't interacted for 1111 days. And sends pending rewards to the targetted user. It's important to highlight that it decrements the totalShares and deletes the user shares

```
        CDP.transfer(_target, pending);
        totalCDPClaimed += pending;
        user.CDPCollected += pending;

        // Reduce the total shares and set the user's shares to 0
        totalShares -= user.shares;
        user.shares = 0;
```

If this is a referred user shares, the same thing happens. However, it doesn't decrement the totalRefShares.

Recommendation

Ensure proper tracking of the totalRefShares variable.

Carpe Diem Team's Comment

totalRefShares keeps track of how many shares are GIVEN to referrers. It doesn't necessarily represent the current amount of shares that referrers have, as it doesn't subtract. This is intended and doesn't cause internal calculation issues as it isn't included in calculations.

Status

Acknowledged

13. Unfixed pragma

Path

CDPToken#L2, Auction.sol#L2, Penson.sol#L2

Description

CDPToken, Auction.sol, Penson.sol is not using fixed solidity versions. Contracts are using unfixed pragma (pragma solidity >=0.8.8), Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version gets selected while deploying a contract which has a higher chances of having bugs in it.

Recommendation

Remove floating pragma and use a specific compiler version with which contracts have been tested.

Status

Resolved

14. CDPToken is not protected for the ERC20 approval race condition

Path

CDPToken.sol

Description

OZ ERC20 implementation is not protected with approve race condition, take a look at this discussion here: **https://github.com/OpenZeppelin/openzeppelin-contracts/issues/4583**

Recommendation

We suggest the team decide whether to add or to not add increaseAllowance, decreaseAllowance on the basis of current business logic.

Additionally, Because the approved spender can frontrun an approve overwrite transaction, enabling him to spend more than what the approver intended. For good practise, users are advised to:

- exactly approve the amount you want the spender to spend.
- in the case of left over open approvals, overwrite the approval to 0 first, then it can be checked that the spender hasn't spent any tokens from previous approval and then the intended approval can be set/overwritten with approve().

Status

Acknowledged

15. Use of payable.transfer() might render contract funds Impossible to Withdraw

Path

Auction.sol#L231

Function

withdrawPLS()

Description

Transfer and send forward a hardcoded amount of gas are **discouraged as gas costs can change**. Also, on certain chains, that cost can be higher than in Mainnet and can result in issues, which led to the issue in the **zkSync Era**.

Recommendation

Use .call() instead.

Status

Resolved

16. Use Check Effects Interaction pattern

Path

Pension.sol

Function

depositCDP(),claimCDP() ,compoundCDP(), Destroyshares()

Description

Currently the CDP token is using ERC20 implementation without hooks. But it can happen that the CDP token might use the implementation with hooks so that the receiver/address can get the callback and can reenter into the transaction flow. For this reason, the CEI pattern can be used.

Follow Check Effect Interaction pattern for depositCDP() claimCDP() compoundCDP() Destroyshares() for L132 burnFrom() L156 transfer() , L194 burn() and L292 transfer().

Do all state changes (variable assignments, increment, decrement) before these external calls.except event emission.

Recommendation

Use CEI pattern as suggested for extra care.

Status

Resolved

17. Use the latest erc20 permit contract implementation

Path

CDPToken.sol#L6

Description

Contract inherited from a draft OpenZeppelin contract. OpenZeppelin contracts may be considered draft contracts if they have not received adequate security auditing or are liable to change with future development.

Recommendation

Use the latest erc20 permit contract implementation. Make sure to use the latest version for all OZ contracts getting used.

Status

Resolved

18. user.lastInteraction can be manipulated/broken

Path

Pension.sol

Function

depositCDP()

Recommendation

- For referral **user.lastInteraction = block.timestamp;** is not getting added while someone is referring to the user while depositing. It can happen that the referral will claim or compound first so the lastInteraction will get set and then deposit something so that the NoUsers will not be increased as it only gets incremented when (user.lastInteraction == 0).

- Any user can call Iamhere() first so that the user.lastInteraction for msg.sender address will get set as block.timestamp (for that current block) and when they deposit for the first time, NoUsers won't get incremented by 1 as the function checks that if (user.lastInteraction == 0) is true then only increment NoUsers variable.

Recommendation

Consider recording lastInteraction for referral in depositCDP(). As stated in 2nd point it is possible for anyone to manipulate user.lastInteraction even before deposit using Iamhere() function, so its important to note that there should not be any logic dependent on the NoUsers.

Status

Resolved

19. Function call can fail because of static length used

Path

Auction.sol

Function

getStatsLoops()

Description

getStatsLoops() function declares return value variables arrays with static size of 10. Because the loop iterates a **numb** amount of time it can happen **numb** amount would be more than 10 so while storing the 11th element it can fail with out of bounds error.

Recommendation

Use dynamic arrays if it is intentional to enter **numb** more than 10. Otherwise, add the require statement which will revert when the **numb** would be greater than 10.

Status

Resolved

20. doDailyUpdate() call tx can go out of gas

Path

Auction.sol

Function

doDailyUpdate()

Description

It should be noted that if the loop exceeds a certain amount of iterations then the transaction can revert with **out of gas** error.

Recommendation

Our recommendation should be to call doDailyUpdate() without skipping any days to ensure that this scenario never happens.

Carpe Diem Team's Comment

This would have quite serious consequences. We agree that the contract should be updated daily. Related to issue #11. We also considered having the pastUpdate logic in a separate function; but this created more problematic scenarios with our current logic. As the userbase grows, the scenario of skipping days would become less and less likely.

Status

Acknowledged

21. More testing should be performed

Description

More test cases should be performed for different scenarios. Writing test cases for different scenarios gives more insights regarding intended and unintended code behaviours.

Recommendation

Perform more robust test cases for different scenarios.

Status

Acknowledged

22. Fix Natspec error

Path

Pension.sol#L250

Description

Natspec comment for the pendingRewardCDP() function returns this error while compiling **Documentation tag @return Pending reward for a given user" does not contain the name of its return parameter.**

Recommendation

Add the return parameter name in natspec comment.

Status

Resolved

23. Check the intended logic in compoundCDP

Path

Pension.sol#L196

Function

compoundCDP()

Description

In compoundCDP() while compounding on L196 the pending amount is getting added to user.CDPCollected but this pending amount is not getting sent to the user as it is getting transferred to the user in claimCDP() and Destroyshares(), it’s getting burned while compounding.

Recommendation

Remove the addition of pending to the user.CDPCollected if it's not intentional while compounding. if it's intentional then add pending to the totalCDPClaimed as it is getting added in claimCDP() and Destroyshares().

Status

Resolved

Informational Issues

24. FEE_DENOMINATOR variable is never used

Path

Auction.sol#L43

Description

The FEE_DENOMINATOR variable is created but never used.

Recommendation

Consider removing unused variable.

Status

Resolved

25. Set minterAddress to immutable

Path

CDPToken.sol#L13

Description

minterAddress variable is set once in the constructor and never changed again elsewhere. Setting this variable to immutable would save a ton of gas as SLOAD would be prevented on runtime.

Recommendation

Set minterAddress as immutable.

Status

Resolved

26. lastUpdate from UserInfo struct is never used

Path
Pension.sol#L32

Description
The variable lastUpdate within the UserInfo struct is declared but remains unused throughout the codebase.

Recommendation
To optimize gas usage and enhance code clarity, it is recommended to remove this unused variable.

Status
Resolved

27. Constructor can be used for variable initialization

Path
Pension.sol#L62-L69

Function
initialize()

Description
Eventually initialize() is calling renounceOwnership() to transfer ownership to address 0 and according to expected logic initialize() should not be used again, which serves the same purpose of the constructor, and also none of this contracts is upgradeable. So for this constructor can be used for variable initialization which will execute while contract deployment and will take arguments.

Recommendation
A constructor can be used for variable initialization.

Carpe Diem Team's Comment
According to the current structure, the three contracts are connected to each other. Until a contract has been deployed, its address is unknown. That's why we need to do this with an initializer function and cannot do it from the constructor.

Status
Acknowledged

28. Time Unit Discrepancy

Path

Pension.sol#L279

Function

Destroyshares()

Description

In the Destroyshares() function at line 279, a duration of 1111 days is used for a condition check. It is crucial to note that the time unit being employed is based on a 24-hour day due to solidity `days` utilized, whereas in the Auction contract, a 20-hour day is utilized.

Recommendation

It is recommended to confirm whether it is intentional to use a 24-hour day for the purpose of checking if shares can be destroyed or not.

Carpe Diem Team's Comment

It is intended behavior.

Status

Acknowledged

29. Confirm the intentional logic

Path

Pension.sol

Function

mintCDP()

Description

mintCDP() is setting **NoUsersPerDay[_day] = NoUsers;** But NoUsers is the variable that keeps count of users interacting with the the pension for the first time so in the above assignment it is assigning a number of users interacted with this contract till the day and not for current **_day**.

Recommendation

Confirm the added logic is intentional.

Carpe Diem Team's Comment

It is intended logic.

Status

Acknowledged

30. Use require instead of if condition

Path

Pension.sol#L80

Function

depositCDP()

Description

if (_amount > 0) {} is used on L80 to execute code block only when the amount is greater than 0. Require statement can be used to revert the transaction in that case.

Recommendation

Use the require statement so that the transaction can be reverted.

Status

Resolved

31. Redundant call to calcDay() in startAuction()

Path

Auction.sol

Function

startAuction()

Description

Redundant call to **calcDay()** in **startAuction()** on L100 to set currentDay. currentDay can be directly set to 0 because calcDay() would be doing **block.timestamp - launchTime** But launchTime would be the current **block.timestamp** so it would be calculating **(block.timestamp-block.timestamp)**.

Recommendation

Set **currentDay** to 0 in startAuction().

Carpe Diem Team's Comment

The first day should be 1, not 0.

1. Added +1 to the calculation in calcDay(), so that the count begins at 1 (and not 0).

2. Call to calcDay() in startAuction() removed and replaced by 1.

Status

Resolved

32. User may opt for cheap alternative to earn CDP

Path

Auction.sol

Description

The user deposits pulsechain native tokens for getting shares so that CDP can be withdrawn for them. if there Pusechain native token to CDP pool available, offering anyone to buy CDP for pulsechain tokens then it can happen that the user will try to buy CDP from there for cheap resulting in less number of people interacting with an auction for buying CDP.

Recommendation

Acknowledge that the team is aware about this scenario.

Carpe Diem Team's Comment

The team is aware about it and it's Intended.

Status

Acknowledged

33. enterAuction() is not updating user.lastInteraction

Path

Auction.sol

Function

enterAuction()

Description

enterAuction() is not updating user.lastInteraction for that user in pension contract.
The user interacts with the protocol using UI then it might happen that he will expect that the user.lastInteraction timestamp should get updated.

Recommendation

Consider verifying the intended logic.

Carpe Diem Team's Comment

We are aware of this but decided to limit it to the Pension contract.

Status

Acknowledged

34. Redundant variable assignment

Path

Auction.sol

Function

calcTokenValue()

Description

else{} on L219 can be removed which is setting **_tokenValue = 0;** on L220 because default value of **_tokenValue** would be 0.

Recommendation

Remove redundant variable assignment.

Status

Resolved

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Carpe Diem codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Carpe Diem smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Carpe Diem smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Carpe Diem to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2023

For

Carpe Diem *Pension*

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com