QuillAudits

Audit Report
September, 2024

For

Table of Content

Table of Content

Table of Content

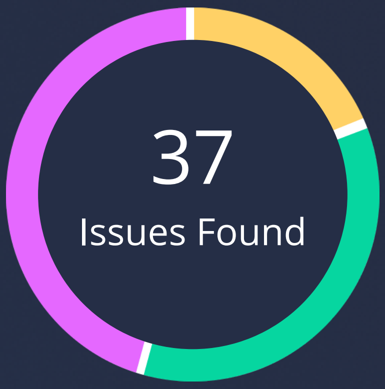
Executive Summary

Project Name	Aconomy
Overview	Aconomy is a decentralized NFT Marketplace for illiquid real-world asset classes to leverage all the monetary benefits that get unlocked when the assets are brought on-chain in the form of PiNFTs. PiNFTs are a special kind of NFTs which possess underlying value in the form of ERC20 tokens(stable coins). The project allows users to sell and buy NFTs using piMarket which also allows auctions. NFTLendingBorrowing contract allows collateralized borrowing/lending functionality. Where the bidder gets the functionality to claim borrower’s NFT incase of the non repayment of the loaned amount. FundingPool can be used to take loans from verified lenders where the pool owner has ability to accept the loans based on the interest of lenders. The poolAddress allows verified users/borrowers to raise a loan request, and verified lenders can then lend the tokens for the request raised.
Timeline	25th June 2024 - 9th July 2024
Updated Code Received	12 August 2024 and 14th August 2024
Second Review	14 August 2024 - 2nd september 2024
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 analyse the Aconomy Codebase for quality, security, and correctness.
Source Code	https://github.com/Pandora-Finance/aconomy-contract/commit/511cf8972daa1623681dbfeba2bbe9165933b764

Executive Summary

Contracts In-Scope	AconomyERC2771Context.sol AconomyFee.sol AttestationRegistry.sol AttestationServices.sol CollectionFactory.sol CollectionMethods.sol FundingPool.sol NFTlendingBorrowing.sol piMarket.sol piNFT.sol piNFTMethods.sol poolAddress.sol poolRegistry.sol poolStorage.sol validatedNFT.sol validatorStake.sol **libraries:-** contracts/Libraries/LibPoolAddress.sol contracts/Libraries/LibNFTLendingBorrowing.sol contracts/Libraries/DateTimeLib.sol contracts/Libraries/LibMarket.sol contracts/Libraries/LibCollection.sol contracts/Libraries/LibCalculations.sol contracts/Libraries/LibPool.sol contracts/Libraries/WadRayMath.sol contracts/utils/LibShare.sol
Branch	commit/511cf8972daa1623681dbfeba2bbe9165933b764
Fixed In	8f4f71e65dafbe2cd936eaf530b4ea8624edac61

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	4	4	10
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	3	9	7

Checked Vulnerabilities

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

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 impact 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.

Medium Severity Issues

1. Denial of service while paying monthly installment

Path

FundingPool.sol, poolAddress.sol

Function

supplyToPool(), repayMonthlyInstallment()
loanRequest(), repayMonthlyInstallment()

Description

When the lender supplies the amount to the pool with supplyToPool() it's getting checked that the _amount is greater or equal to 1000000.

But when the poolOwner tries to repay the monthly amount after accepting the lender's bid, in the repayMonthlyInstallment() on L528 it's getting checked that fundDetail.amount / fundDetail.installment.installments greater or equal to 1000000.

Let's say the lender entered _amount as 1000000 for the supplyToPool(), the check on L184 passed as the _amount is equal to 1000000.

Then while paying the monthly installment on L528 it will revert as in this case fundDetail.amount / fundDetail.installment.installments would be 1e6/6 = 166666, which is less than 1000000. Resulting in denial of service while paying the monthly amount.

While there can be an alternate solution to repay the full amount, the monthly repayment functionality would be useless.

A similar situation occurs in poolAddress.sol with the loanRequest amount being checked to be greater or equal to 1000000, and repayMonthlyInstallment checks that the principal divided by the number of installments is greater or equal to 1000000 (1e6). If the borrower takes a loan worth 1000000 (1e6) tokens, he would be unable to repay the monthly

Recommendation

Consider checking the fundDetail.amount / fundDetail.installment.installments >= 1000000 in the supplyToPool() function itself. if it's less, then the transaction should revert while supplying the tokens.

Status

Acknowledged

Aconomy Team's Comment

The team decides to keep the check, as in this situation instead of paying with repayMonthlyInstallment() the user can pay with RepayFullAmount()/repayFullLoan().

2. Add support for fee-on-transfer tokens.

Path

validatorStake.sol, piNFTMethods.sol, piMarket.sol, NFTlendingBorrowing.sol

Function

_stake(), listNFTforBorrowing(), setExpectedAmount(), Bid()

Description

stake() and addStake() allows anyone to stake any ERC20 token. In case the ERC20 token deducts fees on transfer, the calculation while storing/incrementing the stakedAmount would be incorrect.

E.g. The Validator stakes 100 tokens, the stakedAmount for that validator increased by 100 on L63. But while transferring tokens on L65 only 90 tokens were transferred (as 10 tokens deducted as some tax/fee by token contract). In this case, the increase of 100 on L63 is incorrect. So this can trick the owner into refunding 100 tokens to the validator address even though less than 100 tokens were transferred.

(It should be noted that while transferring the tokens back to the validator with refundStake(), less will be transferred in case of fee deducting tokens.)

In piNFTMethods.addERC20() allows validators to add any ERC20 token. In case the ERC20 token deducts fees on transfer, the calculation in updateERC20() for adding _value in erc20Balances mapping would be incorrect.

There's a similar possibility in both piMarket and NFTlendingBorrowing regarding the usage of fees deducting tokens.
While this issue can be rare, we have decided to highlight the issue as there would be unpredictable possibilities for an open system like this when it comes to the types and incorrect implementations of ERC20 tokens.

Recommendation

Consider storing the contract balance before the token transfer call and after that, it can be subtracted from the current balance value to know exactly the transferred tokens. Later this amount can be used to increment the stake amount in validatorStakes mapping.

Consider storing the contract balance before the token transfer call, and after that it can be subtracted from the current balance value to know the actual transferred token amount. Or restrict the ERC20 token to a specific token address as mentioned in other issue(s).

Status

Acknowledged

Aconomy Team's Comment

We are not using any kind of erc20 token which deducts fees on transfer. We are using a stable coin for staking.

3. Bad debt can accrue over time giving users no incentive to pay back.

Path

NFTLendingBorrowing.sol

Description

A nuanced action from the borrower may be seen when NFT deposits happen, they take loans and then do not pay back the ERC20 token loaned. There is no guarantee that users will pay back the loans they take by returning the ERC20 token along with the interest accrued over time. When enough time passes, retrieving the NFT by repaying the debt would be more expensive than the user could want to bear.

Recommendation

It would be advisable to include a collateral-loan ratio that takes into account the value of the NFTs and the amount bidded for them to determine 'safe' debt levels.

Consider including a cap to which interest can safely be accrued until measures have to be taken against users who have defaulted on repayment.

Status

Resolved

Audit Team's Comment

Aconomy Team created a claimNFT function for the lender to retrieve the NFT if this situation occurs. Fixed here:

https://github.com/Pandora-Finance/aconomy-contract/blob/db24a7b2cb12beafa7c7a08e68db01df05a21d01/contracts/Libraries/LibNFTLendingBorrowing.sol#L102

4. Borrower defaulting is unhandled by the smart contracts

Path

FundingPool.sol, poolAddress.sol

Function

loanRequest(), AcceptLoan()

Description

The isLoanDefaulted() function is currently unused in the codebase, while in an isolated event this wouldn't be an issue but the poolAddress contract has no checks for borrowers who default on loans. As it stands, nothing stands between a defaulting borrower and a new loan being approved - not even with the scrutiny of the proposed lender.

Within the loanRequest() function, there isn't any check/flag that indicates the borrower submitting a loan request has any bad debt. This isn't taken into consideration when the lender calls AcceptLoan(). The borrower's active loans are added to when in the actual sense, previous loans could be past their expiration time, thus defeating the purpose of a liquidation which occurs when liquidators (who are usually incentivized) clear bad debts from the system.

In both FundingPool and poolAddress the precaution needs to be taken if the borrower never returns the taken loan back.

Recommendation

It would be advisable for the AcceptLoan() function to include a check for the borrower's health / credibility and also implement precautionary functionality for cases where defaulters never return the borrowed funds.

Status

Acknowledged

Aconomy Team's Comment

For now we'll be giving this feature to the validator(It's not fixed yet) who will be holding some collateral, so after the loan is defaulted, we can repay that loan with that collateral.

5. The user can choose low to no fees.

Path

piMarket.sol

Function

BuyNFT(), executeBidOrder()

Description

While calling BuyNFT()/executeBidOrder() the user has the ability to set _fromCollection as true/false to choose less (or even more for other users to whom the remaining amount would be sent) fees/royalties amount.

This happens because depending on the if{} or else{} execution , the different royalties are used.

Instead of allowing the user to toggle _fromCollection bool value in BuyNFT() and in executeBidOrder(), another variable can be added to the TokenMeta struct , where the bool value can be set while creating the sale itself.

The contract will make a call on the _contractAddress address entered by the user while creating the sale and will check if it is a collection or not. The ERC-165 standard interface detection can be used for the same.

In both BuyNFT() and in executeBidOrder() this boolean value can be used from the stored TokenMeta and depending on this bool value the royalties would be fetched in if or else.

The implemented functionality should be checked practically as it may break the functionality for contracts that are not supporting the ERC-165 standard interface detection. Additionally the assumption is made that the piMarket is only supporting valid collections/ contract implementations supported by the marketplace as discussed internally.

Recommendation

We recommend the above solution for the mentioned issue and or additionally we encourage the team to think of a more efficient solution, where the manipulating royalty fee is not possible.

Status

Resolved

Resolved In

https://github.com/Pandora-Finance/aconomy-contract/commit/6f1593733df66b193e0881b9d44755ab2c9bd53b

6. User’s NFT can be claimed by another actor by paying the user’s debt at any time

Path

NFTLendingBorrowing.sol

Function

repay()

Description

There is no restriction on who can pay back the loan bidded for by the lender. Any user can repay the loan to receive the NFT that the original lender added as collateral. As it is, there is no time delta between when the user is ‘safe’ to redeem their tokens and when an arbitrageur/liquidator could swoop in to pick up the user’s collateral. This means that users who put up their NFT for sale run the risk of losing their NFT immediately another user repays the amount + interest accrued.

Recommendation

If the user isn’t intended to lose their tokens this way, consider introducing a deadline to adjust when this can happen.

Status

Resolved

Resolved In

https://github.com/Pandora-Finance/aconomy-contract/commit/d2335259c2f771b3a091776c54ae65dbd1386cc7

7. Users can take non collateralized loans

Path

Fundingpool.sol, poolAddress.sol

Description

The users can take non collateralized loans in the Fundingpool and poolAddress. The ability to take non collateralized loans creates the possibility where the borrowers may never return the loans.

Recommendation

Take the on chain collateral from the borrower.

Status

Acknowledged

Low Severity Issues

8. public variables can be marked as private

Path

AconomyFee.sol

Function

AconomyPoolFee(), AconomyPiMarketFee(), AconomyNFTLendBorrowFee()

Description

AconomyPoolFee(), AconomyPiMarketFee(), AconomyNFTLendBorrowFee() are public view methods used to return the values of _AconomyPoolFee,_AconomyPiMarketFee,_AconomyNFTLendBorrowFee variables respectively.

Solidity creates getter methods for every public variable. the contract currently has these above-mentioned functions defined for returning the values, which makes it redundant to have other public methods created by solidity for _AconomyPoolFee,_AconomyPiMarketFee,_AconomyNFTLendBorrowFee.

Recommendation

Consider declaring the _AconomyPoolFee,_AconomyPiMarketFee,_AconomyNFTLendBorrowFee variables as private to avoid the creation of redundant getter functions.

Status

Resolved

9. Owner can refund validator’s stake with any arbitrary ERC20 token, not the staked token

Path

ValidatorStake.sol

Function

refundStake

Description

The contract owner is allowed to perform a refund of the validator’s stake at any time deemed necessary (except when the contract is paused). The validator’s staked tokens could be an arbitrary token, say DAI for example, but the owner is not constrained to refund DAI tokens. They can refund any amount of any ERC20 tokens, which could be worth nothing at all. This way, a malicious/compromised owner account could deny validators access to their staked amounts and keep the staked value stuck in the contract still as there is no central withdraw / sweep function in validatorStake.sol

Recommendation

As suggested in the 5th issue’s recommendation, the validatorStakes mapping can be modified so that the validator address will map to a specific ERC20 address which will map to the StakeDetail like this mapping(address => mapping(address => StakeDetail) public validatorStakes.

By modifying this when the user stakes the specific ERC20 address, the stakedAmount will be increased for that ERC20 address in the validatorStakes mapping and while refunding the amount with refundStake(), the _refundAmount will be subtracted from the previously added stakedAmount. In case the stakedAmount for the ERC20 address for which the refund is getting paid is 0, the transaction will revert because of an arithmetic error. Additionally, the require statement can be used to check if the validator has staked the amount that is getting subtracted from the stakes.stakedAmount on L87 for handling exceptions.

Status

Resolved

Resolved In

https://github.com/Pandora-Finance/aconomy-contract/commit/60a4c4e5eab98a7a62637f3a236fde0ffad11552

10. Redundant variable

Path

validatorStake.sol#L25, AttestationServices#L11

Description

ERC20Address address type variable from StakeDetail struct is not getting used.
The AttestationRegistryAddress variable is not used in any of the contracts.

Recommendation

Consider removing the unused variables in ValidatorStake.sol and AttestationServices.sol

Status

Resolved

Audit Team's Comment

Resolved by Renaming and using the variable.

11. For different _ERC20Address stake the same mapped value is used

Path

validatorStake.sol#L63

Description

When the user stakes the amount, on L63 the amount is getting added in validatorStakes[_validator].stakedAmount. Any ERC20 token can be used by a user while staking.

When a user uses different ERC20 tokens the same mapping is used to add/store the staked amount. This mapping is then used while subtracting an amount while refunding the stake.

This can create misrepresentation for the staked amount, as it’s not possible to understand how much amount of different ERC20 tokens was added by that validator.

Recommendation

Consider maintaining different records for different ERC20 tokens for the specific validator address. Mapping can be modified like this mapping(address => mapping(address => StakeDetail) public validatorStakes where the second address would be the ERC20 address.

Status

Resolved

12. Remove redundant import and it’s usage.

Path

validatorStake.sol, piNFT.sol, ValidatedNFT.sol, CollectionMethods.sol, piMarket.sol, piNFTMethods.sol

Description

- In validatorStake.sol, Counters.sol is getting imported on L8 and L10, and using directive is getting used for it on L19. No functionality from Counters.sol is getting used hence it’s redundant. Consider removing the import on L8,L10 and its usage on L19.
- Same in PoolRegistry.sol, Counters.sol is getting imported on L4 and it’s using directive is on L21. It is also redundant.In piNFT.sol, IERC20 and IERC721Receiver are redundant imports. Consider removing the unused imports.
- IERC20Upgradeable, IERC721ReceiverUpgradeable, Ownable are unused imports in ValidatedNFT.sol and can be removed.
- IERC20, ERC721, LibCollection are unused imports in CollectionMethods.sol and can be removed.
- Solidity pragma versions greater than 0.8.0 have internal underflow and overflow protection. Arithmetic operations that would result in underflow and overflow would result in an internal panic. The SafeMath library helps to protect from such issues in earlier Solidity versions but now becomes redundant as used here.
- In NFTLendingBorrowing.sol, Counters.sol is getting imported on L10 and using directive is getting used for it on L23. No functionality from Counters.sol is getting used hence it’s redundant. Consider removing the import on L10 and its usage on L23.
- ERC721, Ownable are unused imports in PiMarket.sol and can be removed. ERC721 is imported on L5 and Ownable is imported on L6 but they are unused throughout the codebase and can be removed.
- In piNFTMethods the import “./CollectionMethods.sol”; on L12 is unused hence can be removed.
- In poolAddress.sol , the Counters.sol is imported but never used hence can be removed.

Recommendation

Consider removing the redundant imports and declarations as suggested.

Status

Resolved

13. Remove redundant modifier

Path

CollectionMethods.sol

Function

onlyOwnerOfToken

Description

In CollectionMethods.sol, the modifier that checks if a caller is the NFT owner is unused. Although CollectionMethods is imported in the piMarket contract, piMarket.sol defines onlyOwnerOfToken separately, not using the existing function modifier in CollectionMethods.sol.

Recommendation

Consider removing the unused modifier as suggested.

Status

Resolved

14. ValidatedNFT mints tokens without a name and symbol

Path

validatedNFT.sol

Description

The initialize() function in the ValidatedNFT contract does not initialize ERC721Upgradeable by calling __ERC721_init(). The ERC721Upgradeable is inherited in ERC721URIStorageUpgradeable. The __ERC721_init() sets the contract name and symbol in __ERC721_init(). Not calling the initializer function leaves the NFT without a name and symbol which could affect metadata parsing for each ValidatedNFT minted, even though they will have all other information intact.

Recommendation

Call __ERC721_init(_, _) and set the name and symbol as required.

Status

Resolved

15. Code-test functionality mismatch

Path

NFTLendingBorrowing.sol

Function

listNFTforBorrowing(), setExpectedAmount(), Bid()

Description

In listNFTforBorrowing() and setExpectedAmount(), _expectedAmount entered is getting checked to be greater than or equal to 10000000 (1e7) on L174, L238 respectively. Additionally on L264 in Bid(), it checks for require(_bidAmount >= 10000000, “bid amount too low”); which is also 1e7.

Check that _expectedAmount/_bidAmount should be checked to be greater than or equal to 1000000 (i.e. 1e6).

Recommendation

Adjust the code/tests according to the intended functionality.

Status

Resolved

16. Restrict the contract functionalities to specific collections

Path

piNFTMethods.sol, piMarket.sol

Description

In piNFTMethods.sol, since anyone can add the validator for any NFT collection , there's a possibility that collection addresses that are not supported can be also used.

In piMarket.sol, There's similar possibility of using the NFTs from non supported collections for direct sale and auction.

We recommend the project team to decide the appropriate case according to business logic and decide whether or not to allow any NFT collection address. In case only specific addresses/implementations are allowed. The functionality can be restricted to certain addresses.

Recommendation

Check the business requirement and add the restriction for collection addresses if needed.

Status

Acknowledged

17. Add amount 0 restriction in withdraw()

Path

piNFTMethods.sol

Function

withdraw()

Description

Add amount 0 restriction check in withdraw() as allowing 0 amount can create a problem where for the first time the NFT will get transferred and then on L504 the 0 amount will get added to the withdrawnAmount mapping. While calling withdraw() again, on L476 it will check that the msg.sender is owner and it will revert as the NFT is already transferred to the contract while withdrawing the ERC20 first time, even though 0 tokens were withdrawn.

If this type of condition happens then the user again needs to repay 0 amount so the user will get his NFT back.

There are certain situations where this issue can be triggered e.g while making a call on the front end the amount for withdraw ends up to be 0 because of some issues or some unhandled inputs, While this issue can be rare, it's still good practice to add the require check to avoid panic at a critical moment.

Recommendation

Add restriction for the _amount to not equal to 0 in withdraw().

Status

Resolved

18. Usage of any ERC20 token is possible

Path

piNFTMethods.sol, NFTLendingBorrowing.sol, piMarket.sol

Function

sellNFT(), sellNFT_byBid(), Bid(), addERC20()

Description

In piNFTMethods, the added validator can use any ERC20 token while adding ERC20 using addERC20(). Depending on the ERC20 token added this can be the ERC20 token with no value in the real market or the different ERC20 token than what the NFT owner is expecting.

In NFTLendingBorrowing , The lender/bidder can specify the ERC20 token address that they are willing to transfer while bidding. And this is the ERC20 token that gets transferred while accepting the bid without checking if this is the same token that the NFT lister/borrower is willing to borrow.

In piMarket, The seller is allowed to include any ERC20 token as _currency for their NFT sale. This functionality still varies from what is discussed with the team (i.e to allow only specific token e.g USDC to minimize the error).

In FundingPool.sol, Lenders can use any ERC20 token address while lending the amount with supplyToPool().

Recommendation

We recommend restricting the ERC20 address to the specific address in piNFTMethods, NFTLendingBorrowing, piMarket, FundingPool.

Status

Acknowledged

19. lender and borrower verification functionality can be simplified.

Path

AttestationRegistry.sol, AttestationServices.sol

Description

Instead of attestation functionality simple mapping for address to boolean can be used to verify lender and borrower addresses in the poolRegistry. In this way the functionality would be more simple to understand and gas efficient.

Eventually the one who can add the lender and borrower would be the same address i.e. pool owner. In turn it will reduce the attack surface.

Recommendation

Use the mapping to keep track of verified lenders and borrowers.

Status

Acknowledged

20. Remove the redundant functionality

Path

piNFTMethods.sol

Function

addERC20()

Description

In addERC20(), the code checks that the validator is adding ERC20 that is equal to what was added for that collection and token previously,if different then it reverts on L207.

This means the address type variable can be used in the erc20Contracts mapping instead of address array at the last, as it needs to only hold the one address. This in turn reduces other logic like the logic required for swapping and popping the elements in removeERC20().

Additionally, the address of the ERC20 can be hardcoded to simplify the process, instead of the validator passing it.

Recommendation

Map the address instead of address array as suggested.

Status

Acknowledged

Informational Issues

21. Borrower should be aware about the timeframe

Path

NFTlendingBorrowing.sol

Function

ClaimNFT()

Description

There is a possible scenario where the lender adds the bid with short bidDetail.expiration and can claim NFT directly after the block.timestamp goes ahead of bidDetail.expiration. So here the borrower gets less (or say inconsistent) time frame to pay the loan and interest amount.

In this case showing the timeframe on frontend in which the borrower needs to pay the loan would be helpful, so that the borrower would be aware while/before accepting the loan that after the block.timestamp goes ahead of bidDetail.expiration the lender can claim his(borrower's) NFT.

Recommendation

Add a smart contract level logic to add more specific timeframe for all the loans or acknowledge that the user would be able to see the timeframe on the frontend after which the lender can claim the NFT if the borrower fails to repay the loan.

Status

Acknowledged

Acomony Team's Comment

The timestamp for every bid will be shown, so the borrower can see expiration timestamp while accepting the bid.

22. Note regarding the lending and the usage of any NFT in NFTlendingBorrowing

Path

NFTlendingBorrowing.sol

Description

The lenders can check the added amount for specific piNFT for which they are bidding. Doing so helps the lenders to get some assurance that even if the borrower is unable to repay the taken loan+interest, the borrower's NFT can be claimed and this situation can be settled with the validator in real world and or with withdrawing the ERC20 from piNFTMethods.

In the case the borrower is the owner of NFT with no ERC20 added for his NFT in the piNFTMethods, the lenders wouldn't be able to get the assurance that they will get in the case when the borrower would be the user with ERC20 value added to his piNFT. And in the case of the borrower being unable to pay the loan, the lender has no options to settle this situation.

Additionally it should be noted that the NFTlendingBorrowing allows users to list the other NFTs apart from the piNFT.

Recommendation

Check the intended functionality and add the information on frontend and documentation to support the business logic.

Status

Acknowledged

Aconomy Team's Comment

While lending the tokens, the lender would be able to see the information on frontend to decide if the borrower is the user with the ERC20 amount locked for the piNFT. Additionally, In the current version we have decided to keep it open for anyone to decide if they want to lend the tokens to the borrower with no ERC20 locked in piNFTMethods for his NFT.

23. Unused events

Path

Fundingpool.sol, poolAddress.sol

Description

In Fundingpool.sol, InstallmentRepaid() is unused hence can be removed.
In poolAddress.sol, repaidAmounts() event is unused hence can be removed.

Recommendation

Remove unused events.

Status

Resolved

24. Check the intended installment amount that should be returned

Path

FundingPool.sol#L503, poolAddress.sol#L308

Function

viewInstallmentAmount()

Description

In FundingPool on L504 if (monthsSinceStart > lastPaymentCycle) is true then it returns the fundDetail.paymentCycleAmount that would be the amount for the 1 cycle/month but it can happen that more than one months are elapsed after the last repayment in this case the amount needs to be adjusted accordingly.

The same thing is happening in the poolAddress's viewInstallmentAmount().

Recommendation

Check the functionality is according to the requirement.

Status

Acknowledged

Aconomy Team's Comment

This is intended business logic.

25. Note about next due date calculation

Path

FundingPool.sol, poolAddress.sol

Description

In both FundingPool.sol, poolAddress.sol the calculateNextDueDate() calculates the very next due date without taking the current timestamp in account. So it can happen that according to the current timestamp more than 1 due date has elapsed but it will still calculate the very next due date from the last repaid timestamp.

Recommendation

Check the intended functionality.

Status

Acknowledged

26. piNFT.sol, validatedNFT.sol

Path

FundingPool.sol, poolAddress.sol

Description

piNFT and validatedNFT allow functionality to mint NFTs. both piNFT.mintNFT() and validatedNFT.mintValidatedNFT() allows anyone to mint the NFTs.

Assuming that these NFTs represent the on-chain presence of real-world tokens, allowing anyone to create/mint the tokens on-chain through these contracts may affect the business logic.

Recommendation

Consider verifying that the functionality doesn’t affect the business logic.

Status

Acknowledged

Acony Team’s Comment

Until It’s not validated by any whitelisted validator It would not be shown at our platform.

27. A maximum limit for fees can be set.

Path

AconomyFee.sol

Function

setAconomyPoolFee(), setAconomyPiMarketFee(), setAconomyNFTLendBorrowFee()

Description

setAconomyPoolFee(), setAconomyPiMarketFee(), setAconomyNFTLendBorrowFee() functions are setting fees. Currently, there's no maximum limit for fees, the maximum limit for fees can be added to increase the trust among users.

Recommendation

Consider adding a maximum limit for fees.

Status

Resolved

28. The validator's eligibility is not getting checked.

Path

validatorStake.sol, piNFTMethods.sol

Description

While adding the validator with addValidator() and lazyAddValidator(), No check is added in the piNFTMethods for checking if the validator has staked or not.
There's the possibility that the validators that haven't been staked can be also added by users in piNFTMethods.

Recommendation

Consider checking if it is needed to add the code for checking eligible validators based on the stake amount added in validatorStake contract.

Status

Acknowledged

Aconomy Team's Comment

Yes they can do like this but those NFTs would not be visible at our platform and after validating those NFTs they can put it on the marketplace although we will get our fee but there would not be any support by our team.

29. Inadequate error logging in the contracts

Path

CollectionMethods.sol, FundingPool.sol, NFTLendingBorrowing.sol, PiMarket.sol, PiNFTMethods.sol, poolAddress.sol, poolRegistry.sol, ValidatedNFT.sol

Description

As a way of identifying irregularities or where certain requirements are not met, errors are logged in smart contracts. Many of the required statements currently do not have error messages attached to them. Functions could revert without the user knowing/receiving adequate feedback. Adding error messages can also help while handling exceptions on the front end.

Recommendation

Include error messages in the required statements as necessary.

Status

Acknowledged

30. Fully initialize used libraries

Path

ValidatedNFT, CollectionFactory.sol, PiMarket.sol, NFTLendingBorrowing.sol, piNFTMethods.sol, poolAddress.sol

Function

__Pausable_init(), __ReentrancyGuard_init()

Description

The __Pausable_init() calls __Pausable_init_unchained() which sets the _paused to false. Although the default uninitialized value is false as well, it is good practice to call the __Pausable_init within the initialize() call.

The same happens with __ReentrancyGuard_init() which is empty but remains uncalled.

Recommendation

Within the initialize calls of the affected contracts, call the individual *__init() functions as required.

Status

Resolved

31. Require check can be combined

Path

piNFTMethods.sol

Function

addERC20()

Description

On L203, L204 addERC20() function has require checks, The first statement checks that approvedValidator[_collectionAddress][_tokenId] should not be address(0) and the second one checks that msg.sender should be equal to approvedValidator[_collectionAddress][_tokenId].

Instead of checking for both the conditions only one condition can be checked that is require(msg.sender == approvedValidator[_collectionAddress][_tokenId]);.

So if msg.sender == approvedValidator[_collectionAddress][_tokenId] is true then approvedValidator[_collectionAddress][_tokenId] can't be a zero address.

Recommendation

Remove the first check on L203 as suggested in the description.

Status

Resolved

32. Multiple functions yield the same results

Path

PiMarket.sol

Function

getCollectionValidatorRoyalty(), retrieveValidatorRoyalty()

Description

The interfaces within CollectionMethods and PiNFT are similar for validator royalties. In PiMarket.sol, getCollectionValidatorRoyalty() and retrieveValidatorRoyalty() call the getValidatorRoyalties() method on the contract address passed in to obtain the LibShare.Share[] array details.

Recommendation

The two functions can be merged to provide the same results thereby reducing the size of the contracts ABI and a number of function selectors for the compiler to sort through.

Status

Resolved

33. Unused `bidStartTime` in TokenMeta struct

Path

PiMarket.sol

Function

__Pausable_init()

Description

The variable bidStartTime in the TokenMeta struct is declared but never used. It currently takes up an extra portion of storage in every instance of the TokenMeta struct created and will never be filled.

Recommendation

Remove the unused variable.

Status

Resolved

34. Add revert message

Path

piNFTMethods.sol

Description

On L330 it checks that require(erc20Balance >= _value); The revert message can be added.

Recommendation

Add a revert message.

Status

Resolved

35. Users may decide to never return the validator’s withdrawn amount.

Path

piNFTMethods.sol

Description

Users can withdraw the validator’s added ERC20 tokens using withdraw() , on the withdrawal of ERC20 tokens the msg.sender needs to transfer his NFT to the contract address. Which will be transferred back to him when he will repay the withdrawn ERC20 amount.

It may happen that the user decides to not repay the validator’s withdrawn ERC20 amount (and the user never receives the NFT back). This scenario is unpredictable and factors like token values may affect it, But the audit team decided to highlight these scenarios for better assessment of business logic which in turn shares more information to the project team and its users.

Recommendation

Check the mentioned scenario is considered in business logic and users are aware of it.

Status

Acknowledged

Aconomy Team’s Comment

In this case the validator will be having his asset with him.

36. General info

Path

piNFTMethods.sol

Description

Nft is not getting transferred from the NFT owner to the contract when the validator is adding ERC20 with addERC20(). In this case the validator needs to trust the NFT owner, NFT owner may decide to not call redeemOrBurnPiNFT() assuming that the user is already the owner of NFT.

Recommendation

We recommend checking the situation with the business logic and request a comment on this scenario.

Status

Acknowledged

Aconomy Team's Comment

In this case the validator will be having his asset with him.

37. Redundant require statements

Path

PoolRegistry.sol, PoolAddress.sol

Function

_attestAddress(), addLender(), addBorrower(), _repayFullLoan(), repayMonthlyInstalment()

Description

In PoolRegistry.sol#L384, _attestAddress() has a require check that is repeated in the public functions (addLender, addBorrower) that call this internal function. The public functions have the ownsPool modifier which performs the exact same check. This will make the function calls to add lenders and borrowers to the protocol consume more gas than required.

Same happens in poolAddress.sol#L405, where the private _repayFullLoan() function has the same require(loans[_loanId].state == LoanState.ACCEPTED); check as repayMonthlyInstalment().

Recommendation

Consider removing the require check in _attestAddress() require(msg.sender == pools[_poolId].owner, “Not the pool owner”); and the require check in _repayFullLoan() unless these functions are intended to be reused later on in the protocol’s lifecycle / future integrations with other protocols.

Status

Acknowledged

Functional Tests Cases

AconomyFee.sol

- ✓ Only owner should be able to set the pool fee
- ✓ Only owner should be able to set the pi market fee
- ✓ Only owner should be able to set the lend borrow fee
- ✓ Should revert if the caller is not the owner while setting the fees

validatorStake.sol

- ✓ Should be able to stake
- ✓ Should be able to add stake again
- ✓ Only owner should be able to refund stake
- ✓ Only owner should be able to pause and unpause

piNFT.sol

- ✓ Should be able to mint the NFT
- ✓ Should be able to lazy mint the NFT
- ✓ Only NFT owner should be able to burn the NFT
- ✓ Only methods contract should be able to set royalties for validators

validatedNFT.sol

- ✓ Should be able to mint and validate the NFT in the one call
- ✓ Only methods contract should be able to set royalties for validators
- ✓ Only NFT owner should be able to burn the NFT

Functional Tests Cases

CollectionFactory.sol

- ✓ Should be able to create the collection
- ✓ Should be able to set the royalties for the collection
- ✓ Only collection owner should be able to set the royalties
- ✓ Only collection owner should be able to set URI
- ✓ Only collection owner should be able to set name
- ✓ Only collection owner should be able to set symbol
- ✓ Only collection owner should be able to set description

CollectionMethods.sol

- ✓ Only collection owner should be able to mint the NFT
- ✓ Only methods contract should be able to set royalties for validators
- ✓ Only methods contract should be able to delete validator royalties
- ✓ Only NFT owner should be able to burn the NFT

piNFTMethods.sol

- ✓ NFT owner should be able to add the validator
- ✓ The forwarder should be able to add the validator on behalf of NFT owner
- ✓ Approved validator should be able to add the ERC20
- ✓ NFT owner should be able to burn the piNFT
- ✓ NFT owner should be able to redeem added ERC20 tokens
- ✓ NFT owner should be able to withdraw added ERC20
- ✓ NFT owner should be able to repay the withdrawn ERC20

Functional Tests Cases

piMarket.sol

- ✓ Should be able to sell the NFT
- ✓ Should be able to edit the selling price
- ✓ Should be able to buy the NFT
- ✓ Should be able to sell NFT by an auction
- ✓ Should be able to bid for an auction
- ✓ Should be able to accept the auction
- ✓ Should be able to withdraw the bid money
- ✓ Should be able to make the swap request
- ✓ Should be able to cancel the swap request
- ✓ Should be able to accept the swap request
- ✓ Should not be able to bid for the direct sale
- ✓ Should not be able to withdraw the bid money if already withdrawn
- ✓ Should not be able to cancel the swap if msg.sender is not a initiator
- ✓ Only owner should be able to pause and unpause

Functional Tests Cases

NFTlendingBorrowing.sol

- ✓ Should be able to list the NFT for borrowing
- ✓ Should be able to remove the listed NFT
- ✓ Only NFT owner should be able to set percent
- ✓ Only NFT owner should be able to set duration time
- ✓ Only NFT owner should be able to set the expected amount
- ✓ Should be able to bid a loan for a NFT
- ✓ NFT owner should be able to accept the bid
- ✓ NFT owner should be able to reject the bid
- ✓ NFT owner should be able to repay the accepted ERC20 amount + interest to the bidder
- ✓ Bidder should be able to withdraw the bid if bid is not accepted

AttestationRegistry.sol

- ✓ should register schema
- ✓ should revert when a user tries to register the same schema again

AttestationServices.sol

- ✓ should allow self-attestation
- ✓ should revoke by only the attester

Functional Tests Cases

poolRegistry.sol

- ✓ should create pools successfully
- ✓ should correctly set the variables in the poolDetail struct
- ✓ should correctly add and remove lenders and borrowers

FundingPool.sol

- ✗ should repay loans with monthly repayment amount less than 1e6, without repayFullAmount
- ✓ should correctly calculate due dates with unpaid periods considered
- ✓ should repay loans with monthly repayment amount greater than 1e6
- ✓ should increment installments after successful monthly repayments
- ✓ should return accurate installment amounts

poolAddress.sol

- ✗ should repay loans with monthly repayment amount less than 1e6, without repayFullLoan
- ✓ should correctly calculate due dates with unpaid periods considered
- ✓ should repay loans with monthly repayment amount greater than 1e6
- ✓ should increment installments after successful monthly repayments should return accurate installment amounts

poolStorage.sol

- ✓ should not have any functions to alter storage

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 Aconomy codebase. We performed our audit according to the procedure described above.

Some issues of 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 Aconomy 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 Aconomy 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 the Aconomy to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com