QuillAudits

AUDIT REPORT

September 2025

For

iPaza Labs

Table of Content

Executive Summary

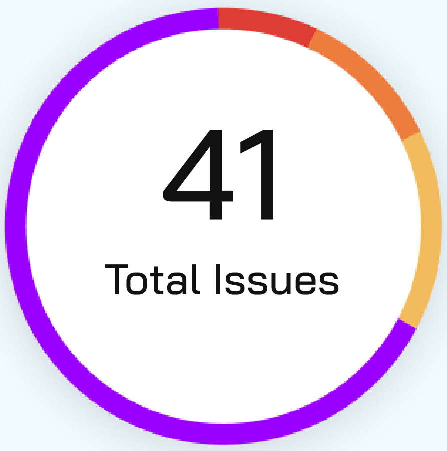
Project Name	iPazaLabs
Protocol Type	DeFi
Project URL	https://pazalabs.io/
Overview	Smart contracts in the codebase facilitate logic for the loan securitization's Defi part at the smart contract level. The Factory helps create/mint loan NFTs and manage the refinancing process with multiple options. It also allows different modes of paying the USDC used for staking and rewards. It also allows whitelisting banks. Every whitelisted bank deploys a pool for itself, which is used for storing staked amounts and rewards and also hosts a logic for transferring stake amounts and rewards. All the critical functions of the contract(s) are protected by access control. LoanNFT is an NFT contract that is used as a loan NFT inside this ecosystem. It is used by contracts like Factory while creating or ingesting loans. Additionally, it allows functionality for batch approvals. PoolOfLoansNFT allows locking loan NFTs inside LockPoolNFT to create a pool of NFTs. It also provides functionality for managing the PoolOfLoansNFT NFTs by freezing, unfreezing, and allowing batch operations for both. LockOutgoingNFT is used for locking an NFT, which is assumed outside of the ecosystem. The system relies on the ERC2771TrustedForwarder to act as a relayer that offsets gas costs to users.
Audit Scope	The scope of this Audit was to analyze the iPazaLabs Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/rahul-ray30/Istakapaza_audit_phase1_SC

Contracts in Scope	• contracts/Factory.sol • contracts/LockPoolNFT.sol • contracts/LoanNFT.sol • contracts/PoolOfLoansNFT.sol • contracts/Pool.sol • contracts/OwnedUpgradeabilityProxy.sol • contracts/LockOutgoingNFT.sol • libraries/ControllerLib.sol • libraries/LoanLib.sol • interfaces/*
Branch	main
Commit Hash	d025c2096d068849633ea3cf861e5a33021ae81f
Language	Solidity
Blockchain	Polygon
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	11th June 2025 - 26th July 2025
Updated Code Received	9th September 2025
Review 2	9th September 2025 - 11th September 2025
Fixed In	2bea958073517e88b6c10bca418ee25ce1472eb1

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



🟥	Critical	0 (0%)
🟥	High	3 (7.30%)
🟧	Medium	4 (9.70%%)
🟨	Low	6 (15.0%)
🟪	Informational	28 (68.0%)

Severity

Issues

	🟥 Critical	🟥 High	🟧 Medium	🟨 Low	🟪 Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	**1**	**11**
Partially Resolved	0	0	0	0	0
Resolved	0	**3**	**4**	**5**	**17**

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Hardcoded Slippage Tolerance in Bonding Curve Operations	High	Resolved
2	Ingested Loan NFTs Frozen Before Validation Check	High	Resolved
3	Vulnerable deadline	High	Resolved
4	Transparent Proxy Pattern Incomplete Implementation	Medium	Resolved
5	Reward Distribution Ignores Individual Loan Tenure	Medium	Resolved
6	Incorrect Reward Transfer Address in Cross-Bank Auctions	Medium	Resolved
7	totalPrincipalAmount can be inflated because of insufficient loan existence check	Medium	Resolved
8	USDC can be transferred with only one function call instead of two	Low	Resolved
9	Only transferFrom() can be used	Low	Resolved
10	setApprovalForAll() can be used instead of other added custom batch approval function	Low	Acknowledged

Issue No.	Issue Title	Severity	Status
11	Redundant require statements in transferFrom() and safeTransferFrom()	Low	Resolved
12	Use only rewardAmount[_loanId] for comparison	Low	Resolved
13	Function definitions from ERC2771ContextUpgradeable can be used	Low	Resolved
14	Hardcoded RPC URL in Configuration	Informational	Resolved
15	Unindexed Event Parameters	Informational	Acknowledged
16	Typographical Errors in Event Names	Informational	Resolved
17	Missing NATSPEC Documentation	Informational	Resolved
18	Inefficient Array Length Caching	Informational	Resolved
19	Inconsistent Parameter Naming	Informational	Resolved
20	Missing Event Emissions for Critical Operations	Informational	Resolved
21	Unused Modifier and Suboptimal Function Visibility	Informational	Resolved
22	Note about upgradeability of PoolOfLoansNFT	Informational	Acknowledged

Issue No.	Issue Title	Severity	Status
23	The trusted forwarder address is hardcoded	Informational	Acknowledged
24	Floating pragma	Informational	Resolved
25	Unused events	Informational	Resolved
26	Note about locked token amount	Informational	Acknowledged
27	The NFT is always owned by contracts or trusted KYCed entities.	Informational	Acknowledged
28	Redundant imports	Informational	Resolved
29	Remove redundant function	Informational	Resolved
30	Check the correct timestamp value	Informational	Acknowledged
31	Unused path in the transferRewards()	Informational	Resolved
32	Note about _serialNo	Informational	Acknowledged
33	Note about lockNFT() access control	Informational	Acknowledged
34	Note about borrowerShare percentage	Informational	Resolved
35	Add comments and maintain documentation	Informational	Acknowledged
36	Remove comments	Informational	Resolved

Issue No.	Issue Title	Severity	Status
37	Unused mapping	Informational	Resolved
38	Note about testcases	Informational	Acknowledged
39	Note about pooled tokens	Informational	Acknowledged
40	Redundant functions in IstakapazaProxyAdmin	Informational	Resolved
41	abicoder v2 declaration can be removed	Informational	Resolved

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls

- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ 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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

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

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

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.

Severity Matrix

Likelihood \ Impact	🟥 High	🟧 Medium	🟨 Low
🟥 **High**	Critical	High	Medium
🟧 **Medium**	High	Medium	Low
🟨 **Low**	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

High Severity Issues

Hardcoded Slippage Tolerance in Bonding Curve Operations

Resolved

Path

contracts/Factory.sol

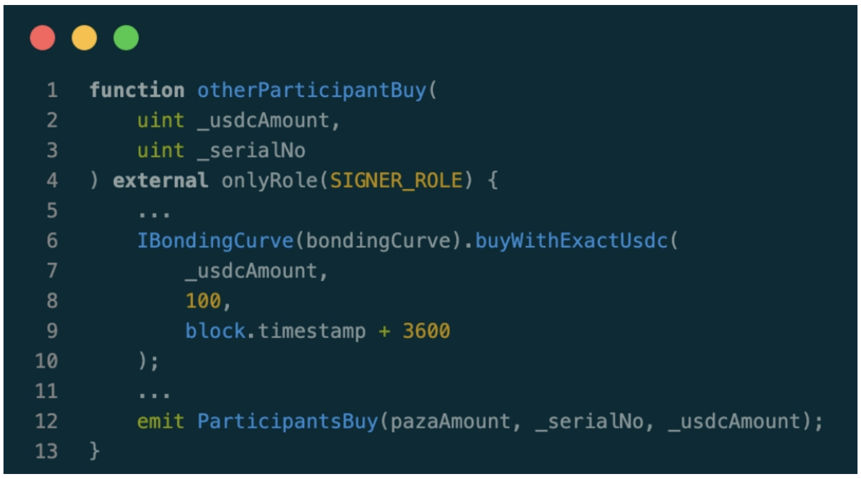
Function

`otherParticipantBuy(), _prepaidPostpaid(), _loanIngestionPrepaidPostpaid(), prepaidPazaBuy()`

Description

Every time when IBondingCurve(bondingCurve).buyWithExactUsdc() is called, the hardcoded slippage value is passed which is 100, without clarifying the units (wei, tokens, or basis points).

The way BondingCurve.buyWithExactUsdc() function checks the slippage tolerance is using **`require(tokenAmount >= _slippageTolerance, "Slippage Tolerance Exceeded");`**



```
function otherParticipantBuy(
    uint _usdcAmount,
    uint _serialNo
) external onlyRole(SIGNER_ROLE) {
    ...
    IBondingCurve(bondingCurve).buyWithExactUsdc(
        _usdcAmount,
        100,
        block.timestamp + 3600
    );
    ...
    emit ParticipantsBuy(pazaAmount, _serialNo, _usdcAmount);
}
```

So here, almost every time, the tokenAmount would be greater than the passed _slippageTolerance that is 100 thereby making the function vulnerable to price manipulations and sandwich attacks.

POC

- The function from Factory.sol calls BondingCurve.buyWithExactUsdc().
- Malicious actor monitors mempool to extract MEV by frontrunning the Factory's buyWithExactUsdc() transaction.
- Trade initiated by the Factory ends with returning lower Paza tokens than that of normal scenario.
- Malicious actor sells his tokens at a premium, making gains from sandwiching the transaction.

Recommendation

BondingCurve.buyCalculateTokenAgainstUsdc() view function can be used off-chain to check the expected amount of the trade. After that, the amount accounting slippage percentage can be calculated, and that amount can be passed to any function in the Factory that calls the BondingCurve.buyWithExactUsdc().

E.g BondingCurve.buyCalculateTokenAgainstUsdc() returns 1e18 off-chain. We are accounting for the 5% of the slippage. So 0.05e18 is 5% of 1e18. So while passing the _slippageTolerance , we can pass (1e18 -0.05e18) = 0.95e18 to the BondingCurve.buyWithExactUsdc(). so that it will check the tokenAmount is greater than the _slippageTolerance provided, otherwise it will revert.

Specific Fixed In Commit

https://github.com/rahul-ray30/Istakapaza_audit_phase1_SC/commit/2bea958073517e88b6c10bca418ee25ce1472eb1

QuillAudits' Response

Currently, it is acknowledged by the protocol team.
There are around 4 instances in the Factory.sol where 95% is calculated as slippage using the amount returned by IBondingCurve(bondingCurve).buyCalculateTokenAgainstUsdc() will remain redundant as highlighted internally.

Update: The most recent change now allows the slippage parameter to be fed directly into the calling function by the off-chain mechanism as described.

Ingested Loan NFTs Frozen Before Validation Check

Resolved

Path

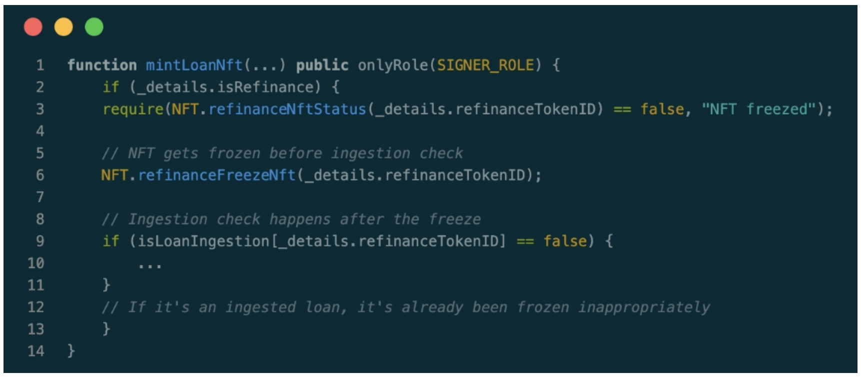
contracts/Factory.sol

Function

`mintLoanNft()`

Description

The protocol freezes loan NFTs during refinancing operations before checking whether they are ingested loans. The refinanceFreezeNft call occurs before the isLoanIngestion check, meaning ingested loans (created via insertNftOnly) can be incorrectly frozen before the system determines they should be exempt from refinancing operations.



```
function mintLoanNft(...) public onlyRole(SIGNER_ROLE) {
    if (_details.isRefinance) {
    require(NFT.refinanceNftStatus(_details.refinanceTokenID) == false, "NFT freezed");

    // NFT gets frozen before ingestion check
    NFT.refinanceFreezeNft(_details.refinanceTokenID);

    // Ingestion check happens after the freeze
    if (isLoanIngestion[_details.refinanceTokenID] == false) {
        ...
    }
    // If it's an ingested loan, it's already been frozen inappropriately
    }
}
```

Impact: High

Ingested loans can be incorrectly frozen before the system validates whether they should participate in refinancing. This disrupts the protocol's loan management system and can lock valuable ingested loan NFTs in incorrect states, breaking the intended separation between regular loans and ingested loans.

Likelihood: Medium

Occurs when ingested loans are targeted for refinancing operations, which could happen through user error or malicious activity.

Recommendation

Move the ingestion validation before the freeze operation.

Vulnerable deadline

Resolved

Path

contracts/Factory.sol

Description

Every time while buying paza tokens with bondingCurve.buyWithExactUsdc() function it passes the deadline as block.timestamp + 3600.

There's no guarantee that the transaction will be executed at the same moment when it's constructed and sent, there can be a delay before the transaction actually executes on the blockchain to be included in the block. resulting in the execution of the trade, where the deadline doesn't actually work anyway.

Recommendation

The deadline should be decided off-chain and should be passed to a function as a parameter, which then will be passed to a bondingCurve.buyWithExactUsdc().

Medium Severity Issues

Transparent Proxy Pattern Incomplete Implementation

Resolved

Path

contracts/OwnedUpgradeabilityProxy.sol

Function

`_fallback()`

Description

The proxy contract doesn't implement the full transparent proxy pattern as recommended by OpenZeppelin. Specifically, it lacks the restriction that prevents admin accounts from calling implementation functions, which could lead to function selector collisions and potential security vulnerabilities.

According to OpenZeppelin documentation, transparent proxies should ensure that admin calls are restricted to upgrade functions only, while regular users can only call implementation functions. The current implementation allows the admin to call all functions, breaking this security model.

TransparentUpgradeableProxy can be used from the OZ instead of using an outdated proxy implementation.

The OZ TransparentUpgradeableProxy will also use the proposed proxy storage slot from ERC-1967.

Additionally, functionality of adding proxy on maintenance (as currently allowed in the present OwnedUpgradeabilityProxy) can be added by overriding OZ's TransparentUpgradeableProxy._fallback().

Recommendation

Use the OpenZeppelin's TransparentUpgradeableProxy as suggested above.

Reward Distribution Ignores Individual Loan Tenure

Resolved

Path

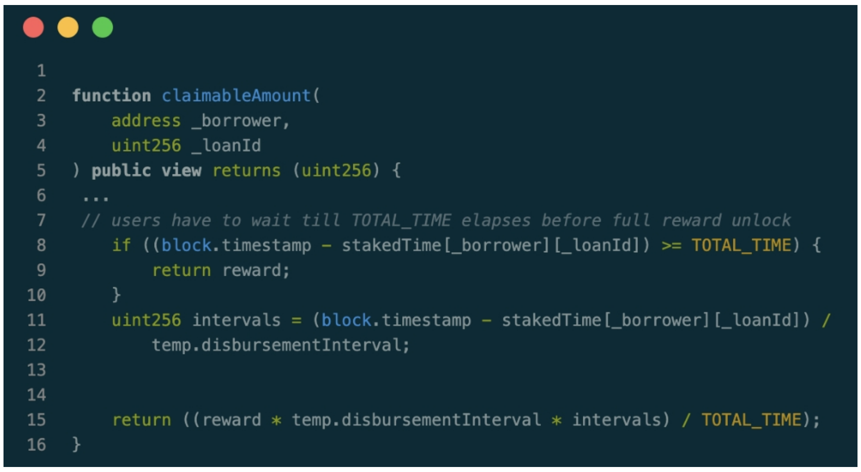
contracts/Pool.sol

Function

`claimableAmount()`

Description

The claimableAmount() function calculates rewards based on a fixed 4-year period (1440 days) regardless of individual loan tenure. Loanswith shorter durations only receive full rewards after the full 4-year period, creating a skewed reward distribution and potential economic inefficiencies.



```

function claimableAmount(
    address _borrower,
    uint256 _loanId
) public view returns (uint256) {
 ...
 // users have to wait till TOTAL_TIME elapses before full reward unlock
    if ((block.timestamp - stakedTime[_borrower][_loanId]) >= TOTAL_TIME) {
        return reward;
    }
    uint256 intervals = (block.timestamp - stakedTime[_borrower][_loanId]) /
        temp.disbursementInterval;


    return ((reward * temp.disbursementInterval * intervals) / TOTAL_TIME);
}
```

Recommendation

Verify the logic should work with TOTAL_TIME or loanTenure.

iPazaLabs team's Response

There is no relation of loanTenure with rewards. The total amount of rewards will be unlocked after 1 year.

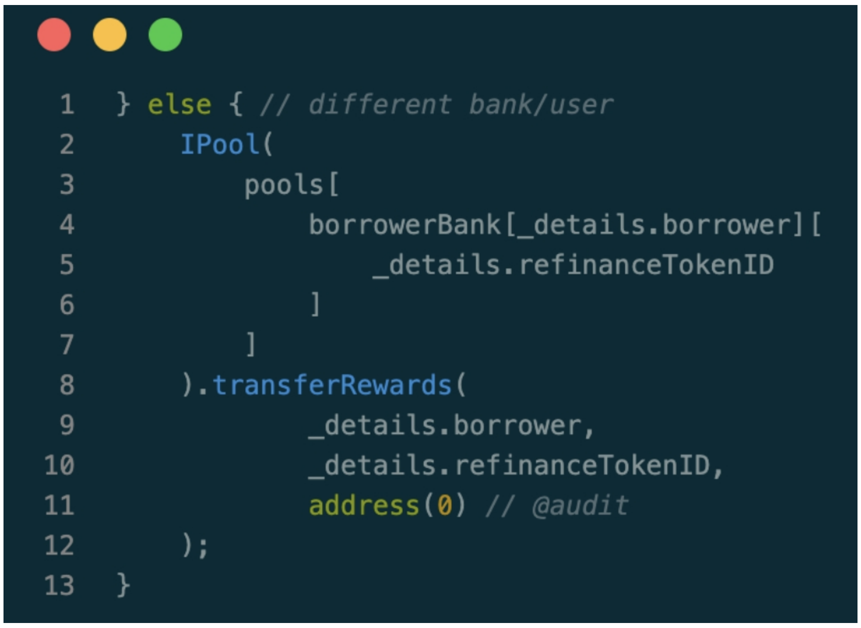
Incorrect Reward Transfer Address in Cross-Bank Auctions

Resolved

Path
contracts/Factory.sol

Function
`mintLoanNft(), transferRewards()`

Description
During refinancing operations, every call to transferRewards() hardcodes address(0) as the auctionWinnerwhich guarantees that the rewards are sent to the bank address. However, in auction scenarios where a different bank or user wins, rewards could end up being sent to the wrong recipient.



```
} else { // different bank/user
    IPool(
        pools[
            borrowerBank[_details.borrower][
                _details.refinanceTokenID
            ]
        ]
    ).transferRewards(
            _details.borrower,
            _details.refinanceTokenID,
            address(0) // @audit
    );
}
```

totalPrincipalAmount can be inflated because of insufficient loan existence check

Resolved

Path

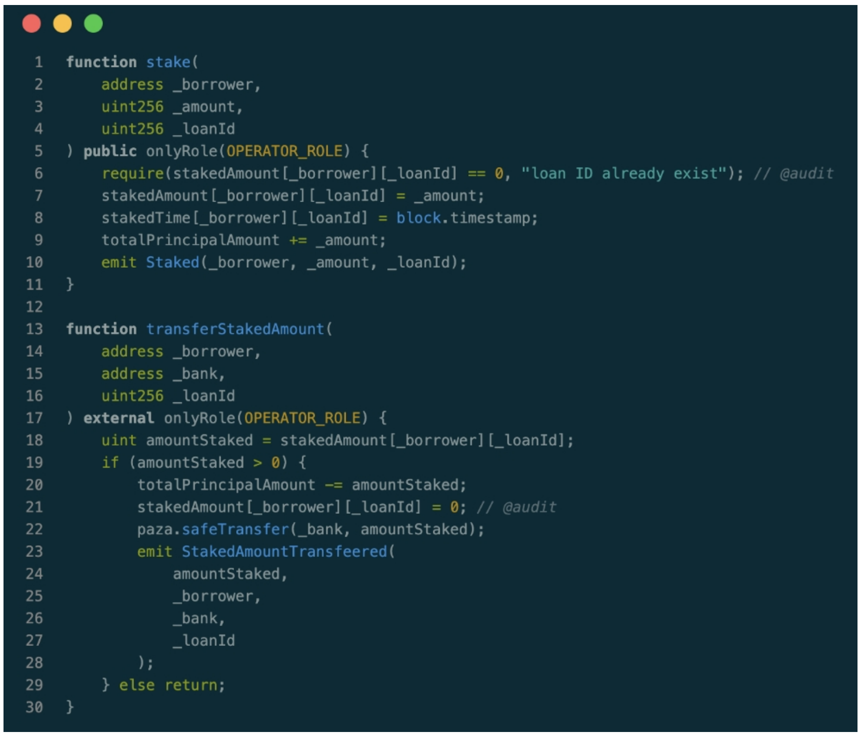
Pool.sol

Function

`transferStakedAmount(), stake()`

Description

After a staked amount is transferred via transferStakedAmount(), the mapping entry defaults to 0, allowing users with OPERATOR_ROLEto stake again on the same loan ID. This can artificially inflate the totalPrincipalAmount and create accounting inconsistencies in the pool’s financial state.



```
function stake(
    address _borrower,
    uint256 _amount,
    uint256 _loanId
) public onlyRole(OPERATOR_ROLE) {
    require(stakedAmount[_borrower][_loanId] == 0, "loan ID already exist"); // @audit
    stakedAmount[_borrower][_loanId] = _amount;
    stakedTime[_borrower][_loanId] = block.timestamp;
    totalPrincipalAmount += _amount;
    emit Staked(_borrower, _amount, _loanId);
}

function transferStakedAmount(
    address _borrower,
    address _bank,
    uint256 _loanId
) external onlyRole(OPERATOR_ROLE) {
    uint amountStaked = stakedAmount[_borrower][_loanId];
    if (amountStaked > 0) {
        totalPrincipalAmount -= amountStaked;
        stakedAmount[_borrower][_loanId] = 0; // @audit
        paza.safeTransfer(_bank, amountStaked);
        emit StakedAmountTransfeered(
            amountStaked,
            _borrower,
            _bank,
            _loanId
        );
    } else return;
}
```

Note

Also note, that this check will be extremely important if the Pool will be implemented outside of the Factory in future releases.

The Factory currently checks that old loans cannot be reused, but the pool does not have this check.

Recommendation

Use another state variable to check for loan existence, e.g. stakedTime. The stakedTime variable is never cleared after it is set through stake().

Low Severity Issues

USDC can be transferred with only one function call instead of two

Resolved

Path

contracts/Factory.sol

Description

prepaidPazaBuy() executes else{} L569 if the usdc.balanceOf(prepaidUsdcWallet)>= usdcAmount is false, it transfers USDC from coinbaseWallet to prepaidUsdcWallet and then from prepaidUsdcWallet to address(this).
Instead, only one transferFrom() call can be used where the USDC can be transferred from the coinbaseWallet to address(this).

Recommendation

Use only one transferFrom() call as suggested above.

Only transferFrom() can be used

Resolved

Path

contracts/Factory.sol

Description

For transferring the tokens from the approved Paza and USDC, in some cases transferFrom() is used, in some cases safeTransferFrom() is used.

Since both the USDC and Paza tokens revert instead of returning the false on failure, only transferFrom() can be used.

Recommendation

Only transferFrom() can be used as suggested.

setApprovalForAll() can be used instead of other added custom batch approval function.

Acknowledged

Path
contracts/Factory.sol

Function Name
`batchApproval(), poolBatchApproval()`

Description
batchApproval(), poolBatchApproval() are used for approving token IDs to the factoryHelper address. These functions are using loops which will use a significant amount of gas.

Recommendation
Instead of using custom functions with loops, setApprovalForAll() from the ERC721 implementation can be used.

iPazaLabs team's Response
While this approach is possible, it would require the NFT holder to approve all NFTs at once. By approving each NFT individually, holders have clear visibility into which NFTs have been approved and which have not, enhancing transparency across the platform.

Redundant require statements in transferFrom() and safeTransferFrom()

Resolved

Path

contracts/LoanNFT.sol

Function Name

`transferFrom(), safeTransferFrom()`

Description

Currently, transferFrom(), safeTransferFrom(address,address,uint256,bytes) are overridden to have the same restrictions as are already in the _transfer().

Since the _transfer() is called in transferFrom(), safeTransferFrom(address,address,uint256) and safeTransferFrom(address,address,uint256,bytes), there's no need to add the same restrictions in these functions by overriding them.

Recommendation

remove overridden transferFrom() and safeTransferFrom(address,address,uint256,bytes).

Use only rewardAmount[_loanId] for comparison

Resolved

Path

contracts/Pool.sol

Function Name

`transferRewards()`

Description

totalPrincipalAmount is not updated in the updateRewardAmount((). But since the rewardAmount[_loanId] is updated in the updateRewardAmount((), only rewardAmount[_loanId] can be used for comparison on L179-L183 in the transferRewards(). while check that the paza.balanceOf(address(this)) is greater than or equal to the reward amount(i.e. rewardAmount[_loanId])

Recommendation

rewardAmount[_loanId] can be used for comparison as suggested.

Function definitions from ERC2771ContextUpgradeable can be used

Resolved

Path

contracts/

Function Name

`_msgSender(), _msgData()`

Description

For both _msgSender(), _msgData() apart from the LockOutgoingNFT contract, all other contracts override and again copy the similar code that is already written in the ERC2771ContextUpgradeable.

Instead of using it again. directly the ERC2771ContextUpgradeable._msgSender() and ERC2771ContextUpgradeable._msgData() can be used as its used in LockOutgoingNFT

These are the contracts in which the changes are required: Factory.sol, LoanNFT.sol, LockPoolNFT.sol, Pool.sol, PoolOfLoansNFT.sol,

Recommendation

Use **ERC2771ContextUpgradeable._msgSender()** and **ERC2771ContextUpgradeable._msgData()** can be used as it is used in LockOutgoingNFT.

Informational Issues

Hardcoded RPC URL in Configuration

Resolved

Path
hardhat.config.ts

Description
The Hardhat configuration contains private RPC URLs that should be moved to environment variables for security and flexibility. If these RPCs are exposed this way even in production, they could be rate-limited causing a DoS, or for RPCs that are paid for, another user could use the credits available on these specific RPC URLs.

Unindexed Event Parameters

Acknowledged

Path
contracts/Pool.sol, contracts/OwnedUpgradeabilityProxy.sol, contracts/Factory.sol

Description
Event parameters should be indexed where appropriate to improve filtering and searchability.

Typographical Errors in Event Names

Resolved

Path
Pool.sol

Function Name
`transferStakedAmount(), updateRewardAmount()`

Description
Several events contain typos: StakedAmountTransfeered should be StakedAmountTransferred, and RewadAmounUpdated should be RewardAmountUpdated.

Missing NATSPEC Documentation

Resolved

Description

Contracts lack comprehensive NATSPEC documentation, which would improve code readability and maintainability.

Inefficient Array Length Caching

Resolved

Path

contracts/LoanNFT.sol, contracts/FactoryHelper.sol

Function Name

`setSharePercent(), poolLoanTransfer(), nftOfNfts(), *batch*(), depositNft()`

Description

Array lengths should be cached to avoid repeated .length calls in loops, improving gas efficiency.

Recommendation

Use a local variable to store the array length when repeated in functions multiple times or called in loops.

Inconsistent Parameter Naming

Resolved

Description

Batch freeze/unfreeze functions use singular parameter names (tokenId) instead of plural (tokenIds) for array parameters, causing confusion.

Missing Event Emissions for Critical Operations

Resolved

Path

contracts/Pool.sol, contracts/Factory.sol

Function Name

`updateAuctionedStakedAmount(), transferAuctionedStakedAmount(), updateAuctionedRewardAmount(), transferAuctionedRewards(), setLockPoolNft(), setBondingcurve(), mintOnlyNft()`

Description

Functions from the following contracts do not emit events:
Critical auction-related functions from Pool.sol : updateAuctionedStakedAmount(), transferAuctionedStakedAmount(), updateAuctionedRewardAmount(),

Functions from Factory.sol : setLockPoolNft(), setBondingcurve()

And from LoanNFT.sol : mintOnlyNft()

This can reduce transparency and make off-chain monitoring difficult.

Recommendation

Consider adding events in the required functions.

Unused Modifier and Suboptimal Function Visibility

Resolved

Path

contracts/LoanNFT.sol, contracts/Pool.sol

Description

The onlyFactoryHelper() modifier is unused, and claimableAmount() function can be marked external instead of public for better gas optimization.

Note about upgradeability of PoolOfLoansNFT

Acknowledged

Path

contracts/PoolOfLoansNFT.sol

Description

Since PoolOfLoansNFT will be deployed in the FactoryHelper.createNewPoolNFT() using EIP 1167 clones, the contract cannot be upgraded.

Recommendation

Consider verifying the mentioned case.

The trusted forwarder address is hardcoded

Acknowledged

Path

contracts/LoanNFT.sol

Description

When the trustedForwarder is passed to the ERC2771ContextUpgradeable contract's constructor, it is hardcoded.

Recommendation

Before moving into the production. It should be validated.

Floating pragma

Resolved

Path
contracts/

Description
Multiple contracts are using version ^0.8.0 or ^0.8.7 with a floating pragma instead of locking to a specific version. Floating pragmas allow the contract to be compiled with any version greater than or equal to the specified version for that major version. If the contract wasn't thoroughly tested with that version, this can introduce possible bugs.

Recommendation
Consider using a fixed solidity version whenever possible.

Unused events

Resolved

Path
contracts/Factory.sol, contracts/LoanNFT.sol, contracts/LockPoolNFT.sol

Description
The following events are unused.
SetTradeContract and MasterPoolLoanNFTSet In Factory.sol
NftFreezed In LoanNFT.sol.
PoolOfLoansNFTContractSet in LockPoolNFT.sol

Recommendation
Remove or use unused events.

Note about locked token amount

Acknowledged

Path

contracts/Factory.sol

Function Name

`mintLoanNft()`

Description

In mintLoanNft() when refinancing with a different lender (else block), the staked amount is locked in the smart contract, and only the reward is transferred to the new lender. According to the intended logic.

The stacked amount remains locked in this case.

Recommendation

Consider verifying the logic.

The NFT is always owned by contracts or trusted KYCed entities.

Acknowledged

Description

This should be noted that most of the functions of this protocol are managed/called by trusted signers. And the LoanNFT NFTs will always be owned by the KYCed/trusted address (in this case, that can be the lender, borrower, investor), and PoolOfLoansNFT NFTs would be owned by the trusted address (in this case, trust).

Recommendation

This note can be acknowledged.

Redundant imports

Resolved

Path

contracts/LoanNFT.sol

Description

In contracts/LoanNFT.sol: AccessControlUpgradeable.sol is imported twice on L8.
In Pool.sol: ILoanNFT.sol and IPool.sol are imported but unused..
In LockPoolNFT.sol: ERC721HolderUpgradeable and OwnableUpgradeable are imported but unused..
In PoolOfLoansNFT.sol: IERC721Receiver is imported but unused.
In LockOutgoingNFT.sol: ContextUpgradeable is imported but unused.

Recommendation

Verify and remove unused imports.

Remove redundant function

Resolved

Path

contracts/LoanNFT.sol

Function Name

`transfer()`

Description

transfer() is redundant, because the tokens will be minted to the bank addresses and transfer() requires the caller should be an address with SIGNER_ROLE role. So in that case, the transfer() becomes redundant.

Recommendation

The redundant transfer() can be r emoved.

Check the correct timestamp value

Acknowledged

Path

contracts/Pool.sol

Description

The comment says the 31104000 is //YEAR IN SECONDS(360 days). Which is correct, as 360 days matches the 31104000.

Still, we recommend checking if the team wants to keep it 360 or 365 days.

Recommendation

Consider checking that the epoch is as expected. In case a change is needed, the timestamp amount for 365 days should be used.

Unused path in the transferRewards()

Resolved

Path

contracts/Pool.sol

Description

Since in the current codebase the pool.transferRewards() is only called from the factory. whenever it is called the _auctionWinner is passed as address(0)

Because of this, the **if{}** block in the transferRewards() is unused.

Recommendation

This can be noted and acknowledged if the **if{}** will be used in the next versions/upgrades.

Note about _serialNo

Acknowledged

Path

contracts/Factory.sol

Description

The _serialNo is not used in any function logic, it is just getting emitted. It is not even checked to be incremented in the smart contract logic. We assume the care will be taken by the caller, as a caller would be an authorized signer.

Recommendation

Consider verifying the logic.

Note about lockNFT() access control

Acknowledged

Path

contracts/PoolOfLoansNFT.sol

Function Name

`lockNFT()`

Description

lockNFT() does not have any access control. since the lockNFT() transfers the NFT to lockPoolNFT. The caller must be the owner of PoolOfLoansNFT NFT.

Since The PoolOfLoansNFT NFT is minted in FactoryHelper.nftOfNfts() to the trust address passed by SIGNER_ROLE. The owner of PoolOfLoansNFT NFT would be a trust address. This shows that the lockNFT() can only be called by the trust address.

Recommendation

The note can be verified and acknowledged.

Note about borrowerShare percentage

Resolved

Path

contracts/Factory.sol

Function Name

`setSharePercent()`

Description

The whitepaper describes the borrowerShare (cashback PAZA), " *As part of an initial special offer, the CeFi platform is offering up to 20% of such PAZAs minted to the borrower, as a reward cashback, which is locked in a staking contract with the lender A.* " as an offering of up to 20% but there are no checks within the contract to ensure that borrowerShare is <= 20%

Recommendation

Include this arithmetic check before borrowerShare/borrowerSharePercent is set.

Add comments and maintain documentation

Acknowledged

Description

Comments for all the critical functions should be added for better readability.

Also, we recommend that the Istakepaza team maintain documentation about how the intended functionality maps to the code written. It will help developers and auditors to understand functionality and will minimize mistakes and confusion in the verification and development of the codebase.

Recommendation

Add comments, maintain documentation, and maintain correct flow in test cases.

Remove comments

Resolved

Path

contracts/interfaces/IFactory.sol, contracts/interfaces/ILoanNFT.sol, contracts/interfaces/IUsdc.sol, contracts/interfaces/IPaza.sol

Description

ILoanNFT.nftStatus() and freezeNFT() are commented.
IUsdc.decimals() and balanceOf() are commented.
In IPaza.sol, the IERC20 import is commented.

Recommendation

Verify and remove comments.

Unused mapping

Resolved

Path

contracts/Factory.sol

Description

Mapping poolLoanNFT, is unused and can be removed.

Recommendation

Remove the unused mapping.

Note about testcases

Acknowledged

Description

We recommend that test cases for the functionality should be checked to reflect the expected flow of function calls. E.g: In current tests stake amount was fetched, which is getting transferred to the pool. As confirmed, it should be the reward amount.

Recommendation

Consider checking the mentioned cases.

Note about pooled tokens

Acknowledged

Path

Factory.sol

Function Name

`nftOfNfts()`

Description

NFTs created in loanNFT are default frozen, they need to be unfrozen to perform actions then are expected to be frozen again, to prevent them from being transferred outside the platform.

When a new pool of loans is created, it does not explicitly lock/freeze the NFTs used to create a new pool token. The system context expects that any tokens not in use and at rest should be frozen to restrict transfers.

Via the FactoryHelper.nftOfNfts() call, the tokens are batch unfrozen to be transferred to the pool contract where they are held, but do not get frozen again when they are at rest.

Recommendation

Can include NFT.batchFreezeNftState(_ids) in the nftOfNfts() call to maintain the NFT state invariant.

Redundant functions in IstakapazaProxyAdmin

Resolved

Path

IstakapazaProxyAdmin.sol

Function Name

`getProxyImplementation(), getProxyAdmin(), changeProxyAdmin(), upgrade(), upgradeAndCall()`

Description

IstakapazaProxyAdmin contract inherits the ProxyAdmin, and the used version of OZ ProxyAdmin already has the below mentioned functions, so overriding these functions is redundant, and these functions can be removed from IstakapazaProxyAdmin:
getProxyImplementation(proxy)
getProxyAdmin(proxy)
changeProxyAdmin(proxy, newAdmin)
upgrade(proxy, implementation)
upgradeAndCall(proxy, implementation, data)

Recommendation

The Redundant functions can be removed.

abicoder v2 declaration can be removed

Resolved

Path

LoanNFT.sol

Description

The usage of pragma abicoder v2 in LoanNFT.sol can be removed since it is activated by default for Solidity v0.8.0.

Recommendation

Remove the pragma abicoder v2 declaration.

Functional Tests

Some of the tests performed are mentioned below:

OwnedUpgradeabilityProxy.sol

- ✓ Only owner should be able to set proxy on maintenance mode
- ✓ Only owner should be able to transfer the proxy ownership
- ✓ Only owner should be able to make calls through proxy when the proxy is in maintenance mode.
- ✓ Only owner should be able to update the proxy implementation.
- ✓ Only owner should be able to update the proxy implementation and make a call on proxy in single call.

Factory.sol

- ✓ refinance with the same bank (with _isNormalRefinance is true in prepaidPazaBuy()).
- ✓ inserted loan refinance with bank WHEN isPrepaid is false.
- ✓ Should not refinance ingested loans when isLoanIngestion() is false.
- ✓ Should transfer rewards to the correct user when refinancing with a different bank/lender.
- ✓ Should send tokens from refinancing to the bank address.
- ✓ Should allow old loans be refinanced.
- ✓ Should revert refinancing when loanTenure has been completed.
- ✓ Should refinance without restrictions on the amount / loanTenure of the new loan.
- ✓ Should transfer stakedAmount and rewards to the same bank when refinancing with the same bank/lender
- ✓ Should transfer auctionStakedAmount and auctionReward to the auction winner when refinancing via an auction

Pool.sol

- ✓ Should be able to update stake amount
- ✓ Should be able to transfer stake amount
- ✓ Should be able to update the reward amount
- ✓ Should be able to transfer the reward amount
- ✓ Should be able to update the auction stake amount
- ✓ Should be able to transfer the auction stake amount
- ✓ Should be able to update the auction reward amount
- ✓ Should be able to transfer the auction reward amount
- ✓ Should prevent re-staking on the same loanId

LoanNFT.sol

- ✓ Should be able to set factory address
- ✓ Should be able to toggle the status of RefinanceFreeze mapping with
- ✓ Should be able to freeze the batch of NFTs
- ✓ Should be able to mint the NFT with setting the borrower and loan details
- ✓ Should be able to mint the NFT only, without setting the borrower and loan details
- ✓ Should be able to approve the NFT
- ✓ Should be able to transfer the approved NFT
- ✓ Should be able to safe-transfer the approved NFT
- ✓ Should be able to batch approve NFTs to factory address
- ✓ Should be able to batch approve NFTs to spender address
- ✓ Owner should be able to burn the token id.

LockPoolNFT.sol

- ✓ Should return NFT details
- ✓ Should allow only poolOfLoans contract to set NFT details
- ✓ Should allow only FactoryHelper whitelist poolOfLoans contract

Threat Model

Contract	Function	Threats
LoanNFT.sol	initialize	Failure in role initialization
	mint	Unexpected state updates, Unexpected token URI updates, Unauthorized external access
	mintOnlyNft	Unexpected state updates, Unexpected token URI updates, Unauthorized external access
	batchApproval	Missing length checks, Unexpected allowance updates
	poolBatchApproval	Missing length checks, Unexpected allowance updates
	batchApprovalForOutgoingNFT	Missing length checks, Unexpected allowance updates
Factory.sol	initialize	Failure in role initialization, Incorrect variable initialization
	mintLoanNft	Unexpected NFT minting, staked amount flow, defi/cefi share flow, rewards flow, Unauthorized external access

Contract	Function	Threats
	prepaidPazaBuy	Unexpected updates in the reward amount, Unexpected updates to the auction reward amount, incorrect transfers of USDC, Incorrect conversions of paza for distribution, Unauthorized external access.
	whitelistBank	Incorrect whitelisting, Unauthorized external access
	completeTenure	Failure to freeze NFT
	insertNftOnly	Unexpected NFT minting, defi/cefi share flow, Unauthorized external access
PoolOfLoansNFT.sol	initialize	Failure in role initialization, Incorrect variable initialization
	lockNFT	Unexpected state updates, Failure to lock loan NFT by sending to LockPoolNFT
	freezeNft	Failure to freeze NFT
	unFreezeNft	Failure to unfreeze the correct NFT
	batchFreezeNft	Failure to batch freeze NFTs
	batchUnFreezeNft	Failure to batch unfreeze NFTs
	mint	Failure to mint and freeze the PoolOfLoansNFT NFT
Pool.sol	initialize	Failure in role initialization, Incorrect variable initialization

Contract	Function	Threats
	claimableAmount	Failure in handling less or more elapsed time than TOTAL_TIME, or arithmetic errors in claimable amount calculations
	stake	Failure in stake amount verification, Unexpected state updates, Unauthorized external access
	updateAuctionedStakedAmount	Unexpected state updates, Unauthorized external access
	transferAuctionedStakedAmount	Failure in transfer amount verification, Unauthorized external access
	updateAuctionedRewardAmount	Unexpected state updates, Unauthorized external access
	transferAuctionedRewards	Failure in transfer amount verification, Unexpected state updates, Unauthorized external access
	updateRewardAmount	Unexpected state updates, Unauthorized external access
	transferRewards	Failure in transfer amount verification, Unexpected state updates, Unauthorized external access
LockPoolNFT.sol	initialize	Failure in role initialization
	updateNFTDetails	Failure in state updates, Unauthorized external access

Contract	Function	Threats
	whitelistPoolOfLoansNft Contract	Failure to whitelist the contract, Unauthorized external access
LockOutgoingNFT.sol	initialize	Failure in role initialization. Incorrect variable initialization
	depositNFT	Length check problems, Failure to lock NFT in the contract, and backdoors for NFT transfers.
	setAllowedNFTAddress	Unauthorized external access
OwnedUpgradeability Proxy	setMaintenance	Failure to enable maintenance mode, Unauthorized external access
	upgradeTo	Failure to upgrade, Unauthorized external access
	upgradeToAndCall	Failure to upgrade and initiate, Unauthorized external access
	fallback	Failure in delegating calls to implementation.

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.

Trust & assumptions / Notes to the user

Centralization risk (backend service): In the codebase, majority of the functions are called via a backend service that is expected to properly format input and provide the correct addresses for fees, refunds, tokens to be sent to.
It is assumed that this backend service properly checks the validity of loans passed into the ecosystem. This backend service is outside the scope of the audit, and is assumed to be trusted to provide correct data in every scenario.

Closing Summary

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

Issues of High, Medium, Low, and Informational severity were found. iPazaLabs team resolved few and acknowledged other issues

Disclaimer

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 seven years of expertise, we've secured over 1400 projects globally, averting over $3 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.

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

September 2025

For

iPaza Labs

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com