Principal Protocol MVP

Smart Contract Security Audit

Prepared by Nexarias

March 18th, 2025 - March 25th, 2025

Nexarias.com

contact@nexarias.com

Document Properties

Client	Principal Protocol
Version	0.1
Classification	Public

Scope

The Principal Protocol MVP smart contracts (Audit and Re-audit smart contracts)

Link	Commit
https://github.com/danushkaadikari/PrincipalProtocol	3d43283b2dc4a7eff98d71e5749c26378e5d613a

Scope

Link	Commit
https://github.com/danushkaadikari/PrincipalProtocol	9624c7604545c60c38dc00e21cefeb530ea02b4d

Contacts

COMPANY	CONTACT
Nexarias	contact@nexarias.com

Contents

1 Introduction

Principal Protocol engaged Nexarias to conduct a security assessment on the Principal Protocol MVP beginning on March 18th, 2025 and ending March 25th, 2025. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Principal Protocol MVP

A decentralized platform for real estate project funding through NFTs. Users can participate in funding real estate projects by purchasing NFTs of their chosen projects.

Issuer	Principal Protocol
Website	https://prnpl.io
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

Nexarias used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by Nexarias are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

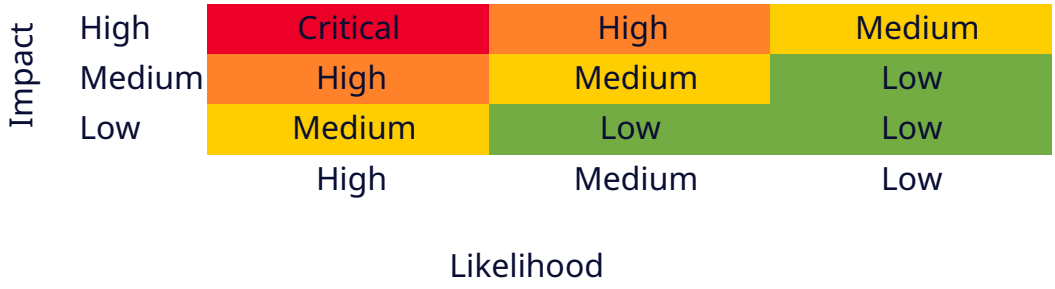
Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.

- Impact quantifies the technical and economic costs of a successful attack.

- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Principal Protocol MVP implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 2 critical-severity, 2 high-severity, 6 medium-severity, 13 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Incorrect Contract Target in handleDefault Function	CRITICAL	Fixed
Missing Check for NFT Contract Validity	CRITICAL	Fixed
Inconsistent Dual Borrow Rate Calculations in InterestRateModel	HIGH	Fixed
Missing Access Control for handleDefault Function	HIGH	Fixed
Incorrect Compound Interest Simulation with 3x Multiplier in calculateInterest	MEDIUM	Fixed
Unauthorized Access to Admin Functions	MEDIUM	Acknowledged
Lack of Slippage Protection in repay	MEDIUM	Fixed
Potential Denial of Service in lockNFTCollateral	MEDIUM	Fixed
Missing Value Checks for pricePerNFT and maxSupply in createCollection	MEDIUM	Fixed
Gas Inefficiency in setPaused Loop	MEDIUM	Fixed
Misleading Rate Scaling Comments in calculateBorrowRate and calculateLendRate	LOW	Fixed

Floating Pragma Version	LOW	Acknowledged
Lack of Input Validation in Setter Functions	LOW	Fixed
Missing Event Emission in Some Functions	LOW	Fixed
Hardcoded Initial Values in Constructor	LOW	Acknowledged
Unrestricted Pause State Modification	LOW	Fixed
Missing Event for Pause State Change	LOW	Fixed
No Collection Removal or Invalidation Mechanism	LOW	Fixed
Missing Validation of Constructor Inputs	LOW	Fixed
No Maximum Admin Limit	LOW	Fixed
Single-Step Ownership Transfer	LOW	Fixed
Gas Inefficiency in Array Removal	LOW	Fixed
Floating Pragma Version	LOW	Acknowledged

3 Finding Details

A InterestRateModel.sol

A.1 Inconsistent Dual Borrow Rate Calculations in InterestRateModel [HIGH]

Description:

The InterestRateModel contract contains two separate functions for calculating borrow rates, creating potential confusion and inconsistency. The public calculateBorrowRate function starts with a 3% base rate and increases to a maximum of 15% at 100% utilization. In contrast, the internal _calculateBorrowRate function, used for lending rate calculations, starts at a 5% base rate and can rise to 100% at full utilization. This duality raises concerns: (1) if the system (e.g., LiquidityHub) uses the public function for borrowing and the internal one for lending, rates may misalign, potentially leading to lending rates exceeding borrowing rates, defying financial logic; (2) it's unclear which function is the intended standard, causing uncertainty. This could lead to incorrect rate applications, affecting economic balance and user trust.

Listing 1: InterestRateModel.sol

```
87 function calculateBorrowRate(uint256 utilization) external pure returns
       ,→ (uint256) {
88     uint256 baseRate = 300; // Base rate: 3% (300 basis points)
89     if (utilization <= 8000) { // Up to 80%
90         return baseRate + ((utilization * 500) / 8000); // Linear to 8%
91     } else {
92         uint256 excess = utilization - 8000;
93         return 800 + ((excess * 700) / 2000); // Exponential to 15%
94     }
95 }

97 function _calculateBorrowRate(uint256 utilization) internal pure returns
       ,→ (uint256) {
```

```
98      uint256 baseRate = 500; // Base rate: 5% APY
99      uint256 optimalUtilization = 8000;
100     if (utilization <= optimalUtilization) {
101         uint256 slope = ((2000 - baseRate) * BASIS_POINTS) /
            ,→ optimalUtilization;
102         return baseRate + ((slope * utilization) / BASIS_POINTS);
103     } else {
104         uint256 excessUtilization = utilization - optimalUtilization;
105         uint256 slope = ((10000 - 2000) * BASIS_POINTS) / (BASIS_POINTS -
            ,→ optimalUtilization);
106         return 2000 + ((slope * excessUtilization) / BASIS_POINTS);
107     }
108 }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Unify the borrow rate calculation by combining the two functions into a single, consistent model that serves both borrowing and lending purposes. This ensures predictable and logically related rates across the system, enhancing clarity and reliability.

Status - Fixed

A.2 Incorrect Compound Interest Simulation with 3x Multiplier in calculateInterest [MEDIUM]

Description:

The calculateInterest function uses a 3x multiplier (labeled 3x APY effect") to simulate compound interest without exponentiation. However, this does not accurately replicate true compound interest (e.g.,

$principal * (1 + apy/10000)^{secondsPassed/SECONDS\_PER\_YEAR} - principal$). For short periods (e.g., 0.5 years at 10% APY), the effect is minimal (50 vs. 48.8); for 1 year, it aligns with simple interest (100 vs. 100); for longer periods (e.g., 2 years), it overshoots (400 vs. 210). The approach misleads users expecting standard compounding and could cause economic imbalances.

Listing 2: InterestRateModel.sol

```
25 function calculateInterest(uint256 principal, uint256 lastUpdateTime,
       ,→ uint256 currentTime, uint256 apy) external pure returns (uint256)
       ,→ {
26     if (currentTime <= lastUpdateTime  principal == 0  apy == 0) return
           ,→ 0;
27     uint256 secondsPassed = currentTime - lastUpdateTime;
28     uint256 principalTimesAPY = (principal * apy);
29     uint256 interest = (principalTimesAPY * secondsPassed) / (
           ,→ SECONDS_PER_YEAR * BASIS_POINTS);
30     uint256 multiplier = (secondsPassed * apy * 3) / SECONDS_PER_YEAR +
           ,→ BASIS_POINTS; // 3x APY effect
31     interest = (interest * multiplier) / BASIS_POINTS;
32     if (secondsPassed > SECONDS_PER_YEAR / 4) {
33         uint256 yearsFactor = (secondsPassed * BASIS_POINTS) /
               ,→ SECONDS_PER_YEAR;
34         interest = (interest * (BASIS_POINTS + yearsFactor / 2)) /
               ,→ BASIS_POINTS;
35     }
36     if (apy > BASIS_POINTS) {
37         uint256 yearsFactor = (secondsPassed * BASIS_POINTS) /
               ,→ SECONDS_PER_YEAR;
38         uint256 apyFactor = (apy * 2) / BASIS_POINTS;
39         interest = (interest * (BASIS_POINTS + apyFactor * yearsFactor /
               ,→ 2)) / BASIS_POINTS;
40     }
41     return interest;
42 }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Option 1: Replace with accurate compound interest using a trusted library for precise exponentiation.Option 2Simplify to periodic compounding (e.g., daily intervals).Option 3:Retain custom logic but document clearly that it uses a non-standard growth model, not true compounding.

Status - Fixed

A.3 Misleading Rate Scaling Comments in calculateBorrowRate and calculateLendRate [LOW]

Description:

Comments in calculateBorrowRate and calculateLendRate claim that returned rates are scaled by 1e18" (a common DeFi convention where 1e18 = 100%), but the functions return values in basis points (e.g., 800 for 8%, 1500 for 15%). In typical DeFi systems, a 10% rate scaled to 1e18 would be 0.1 * 1e18, not 1000 basis points as implemented here.This mismatch between comments and behavior could confuse developers or users.

Listing 3: InterestRateModel.sol

```
82  /**
83   * @notice Placeholder for future dynamic borrow rate calculation
84   * @param utilization The current utilization rate
85   * @return The borrow rate scaled by 1e18
86   */
87  function calculateBorrowRate(


89  /**
90   * @notice Placeholder for future dynamic lending rate calculation
91   * @param utilization The current utilization rate
```

```
92    * @return The lending rate scaled by 1e18
93    */
94   function calculateLendRate(
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Option 1: Update comments to accurately state that the functions return rates in basis points (where 10000 = 100%), reflecting the current implementation. Option 2: Modify the functions to return rates scaled to 1e18 (e.g., multiply basis points by 1e16), aligning with the comments and common DeFi practices.

Status - Fixed

A.4 Floating Pragma Version [LOW]

Description:

The smart contract uses a floating Solidity pragma (0.8.20). This means the compiler may use any minor version within the 0.8.x range, potentially leading to unexpected behavior if a newer minor version introduces changes or deprecations. Using a fixed Solidity version enhances security by ensuring consistency in compilation.

Listing 4: InterestRateModel.sol

```
2   pragma solidity ^0.8.20;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Specify a fixed Solidity version (e.g., pragma solidity 0.8.20;) to ensure consistent compilation and avoid potential issues from unintended compiler updates. Always verify that the chosen version is the most suitable for security and functionality.

Status - Acknowledged

B LiquidityHubAdmin.sol

B.1 Incorrect Contract Target in handleDefault Function [CRITICAL]

Description:

In the LiquidityHubAdmin contract, the handleDefault function uses ILiquidityHub(address(this)) to call setNFTLockStatus and clearBorrowerPosition, targeting the LiquidityHubAdmin address instead of the intended LiquidityHub contract. Since LiquidityHubAdmin does not implement the ILiquidityHub interface or manage borrower positions and NFTs, these calls revert at runtime, preventing liquidation of defaulted loans and undermining default management.

Listing 5: LiquidityHubAdmin.sol

```
132 function handleDefault(address borrower, address nftContract) external
        ,→ nonReentrant onlyAdmin {
133     // Get borrower position from main contract
134     (uint256[] memory collateralNFTs, uint256 borrowedAmount, uint256
            ,→ lastUpdateTime, uint256 totalCollateralValue) =
135         ILiquidityHub(address(this)).borrowerPositions(borrower);

137     // Check if there is an outstanding loan with collateral
138     require(borrowedAmount > 0, "No outstanding loan");
139     require(collateralNFTs.length > 0, "No collateral");

141     // Calculate current loan health
```

```
142         uint256 interest = interestRateModel.calculateInterest(
143             borrowedAmount,
144             lastUpdateTime,
145             block.timestamp,
146             borrowingAPY
147         );
149         uint256 totalOwed = borrowedAmount + interest;
150         uint256 loanHealth = (totalCollateralValue * 10000) / totalOwed; //
                ,→ Using basis points (10000 = 100%)
152         require(loanHealth <= defaultThreshold, "Not in default");
154         // Transfer all NFTs to treasury and update state
155         for (uint256 i = 0; i < collateralNFTs.length; i++) {
156             uint256 tokenId = collateralNFTs[i];
157             // Update NFT lock status in main contract
158             ILiquidityHub(address(this)).setNFTLockStatus(nftContract,
                    ,→ tokenId, false);
159             IERC721(nftContract).safeTransferFrom(
160                 address(this),
161                 treasuryWallet,
162                 tokenId
163             );
164         }
166         // Clear borrower position in main contract
167         ILiquidityHub(address(this)).clearBorrowerPosition(borrower);
169         emit LoanDefaulted(borrower, collateralNFTs);
170     }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Store the LiquidityHub contract address in LiquidityHubAdmin (e.g., via constructor or setter) and update the handleDefault function to call ILiquidityHub(liquidityHubAddress) instead of ILiquidityHub(address(this)) to correctly target the contract managing borrower positions and NFT custody.

Status - Fixed

B.2 Unauthorized Access to Admin Functions [MEDIUM]

Description:

The onlyAdmin modifier relies on owner() == msg.sender, utilizing the Ownable contract's ownership model. However, Ownable permits ownership transfer, posing a risk if the owner account is compromised or ownership is inadvertently transferred to an untrusted party, granting them full control over critical functions.

Listing 6: LiquidityHubAdmin.sol

```
72 modifier onlyAdmin() {
73     require(owner() == msg.sender, "Not authorized");
74     _;
75 }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Replace single-owner control with a multi-signature wallet or DAO-based governance system for critical administrative actions to mitigate the risk of a single point of failure.

Status - Acknowledged

B.3 Lack of Input Validation in Setter Functions [LOW]

Description:

Setter functions such as setLendingAPY, setBorrowingAPY,setWithdrawalFee, and setBorrowingLimit enforce upper limits but lack lower-bound checks. For example, setting lendingAPY or borrowingAPY to 0 could disrupt the protocol's economic incentives, making lending or borrowing unprofitable.

Listing 7: LiquidityHubAdmin.sol

```
86 function setLendingAPY(uint256 newAPY) external onlyAdmin {
87     require(newAPY <= 20000, "APY too high"); // Max 200% (20000 basis
       ,→ points)
```

Listing 8: LiquidityHubAdmin.sol

```
96 function setBorrowingAPY(uint256 newAPY) external onlyAdmin {
97     require(newAPY <= 200000000, "APY too high"); // Max 2000000%
       ,→ (200000000 basis points)
```

Listing 9: LiquidityHubAdmin.sol

```
106 function setWithdrawalFee(uint256 newFee) external onlyAdmin {
107     require(newFee <= 10000, "Fee too high"); // Max 100% (10000 basis
       ,→ points)
```

Listing 10: LiquidityHubAdmin.sol

```
116 function setBorrowingLimit(uint256 newLimit) external onlyAdmin {
117     require(newLimit <= 8000, "Limit too high"); // Max 80% (8000 basis
       ,→ points)
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Introduce minimum value checks for all configuration parameters to ensure they remain within a functional range that supports the protocol's intended operation.

Status - Fixed

B.4 Missing Event Emission in Some Functions [LOW]

Description:

The toggleEmergencyPause function alters critical contract state but does not emit events. This lack of transparency hinders off-chain monitoring and auditing of administrative actions, reducing trust in the system.

Listing 11: LiquidityHubAdmin.sol

```
172 function toggleEmergencyPause() external onlyAdmin {
173     if (paused()) {
174         _unpause();
175     } else {
176         _pause();
177     }
178 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Add event emissions to all state-changing functions to enhance transparency and enable better tracking of administrative changes.

Status - Fixed

B.5 Hardcoded Initial Values in Constructor [LOW]

Description:

The constructor sets fixed initial values for parameters like borrowingLimit (4000 basis points), lendingAPY (300 basis points),and borrowingAPY (600 basis points).Changing these requires redeployment, limiting flexibility if business requirements evolve.

Listing 12: LiquidityHubAdmin.sol

```
58 constructor(address _interestRateModel) {
59     require(_interestRateModel != address(0), "Invalid interest rate
       ,→ model");
60     interestRateModel = IInterestRateModel(_interestRateModel);

62     // Initialize with default values
63     borrowingLimit = 4000; // 40% (4000 basis points)
64     defaultThreshold = 5000; // 50% (5000 basis points)
65     lendingAPY = 300; // 3% (300 basis points)
66     borrowingAPY = 600; // 6% (600 basis points)
67     withdrawalFee = 0; // 0% initially
68     useDynamicRates = false; // Use fixed rates by default
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Allow initial configuration via constructor parameters or post-deployment setter functions with proper access controls to improve adaptability.

Status - Acknowledged

C LiquidityHub.sol

C.1 Missing Check for NFT Contract Validity [CRITICAL]

Description:

In lockNFTCollateral, the nftContract address is not verified as a valid or trusted ERC721 contract supporting IRealEstateNFT. A malicious contract mimicking these interfaces can fake ownership, simulate transfers without moving assets, and inflate collateral values via getCollectionInfo, enabling attackers to borrow excessive funds against worthless collateral. This exposes the protocol to significant financial loss without securing real assets. The RealEstateNFTFactory provides a mechanism to identify legitimate collectionsbut this is not utilized,leaving the function vulnerable to untrusted inputs.

Listing 13: LiquidityHub.sol

```
207 function lockNFTCollateral(
208         CollateralInput[] calldata inputs
209 ) external nonReentrant whenNotPaused {
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Verify that nftContract is a valid collection created by the RealEstateNFTFactory by checking the isValidCollection mapping before processing transfers and value

calculations. This ensures only trusted contracts are accepted, preventing malicious exploits and guaranteeing collateral integrity.

Status - Fixed

C.2 Missing Access Control for handleDefault Function [HIGH]

Description:

The handleDefault function in LiquidityHub allows anyone to liquidate a borrower's position when the total owed amount (principal plus interest) exceeds a threshold relative to the maximum borrowable value based on collateral. Without an access control modifier, unauthorized parties can call this function as soon as the default condition is met, potentially leading to premature or malicious liquidations that harm borrowers or disrupt protocol stability.This deviates from typical lending protocols where liquidation is restricted to authorized liquidators or incentivized roles, undermining operational control over default enforcement.

Listing 14: LiquidityHub.sol

```
504 function handleDefault(address borrower) external {
505     BorrowerPosition storage position = borrowerPositions[borrower];
506     require(position.borrowedAmount > 0, "No outstanding loan");
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Restrict access to the handleDefault function by adding a modifier that limits execution to the admin or a designated liquidator role, ensuring only trusted entities can trigger liquidations and aligning the function with secure protocol governance practices.

Status - Fixed

C.3 Lack of Slippage Protection in repay [MEDIUM]

Description:

The repay function allows borrowers to overpay beyond their total owed amount (principal + interest) without refunding the excess. This could lead to unintentional fund loss if a borrower miscalculates or interest accrues during transaction confirmation.

Listing 15: LiquidityHub.sol

```
299 function repay(uint256 amount) external nonReentrant {
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Implement a refund mechanism for excess payments or restrict repayment amounts to the exact total owed to protect borrowers from overpayment.

Status - Fixed

C.4 Potential Denial of Service in lockNFTCollateral [MEDIUM]

Description:

The lockNFTCollateral function processes multiple NFTs in a loop. A borrower locking a large number of NFTs in one transaction could hit block gas limits, causing transaction failures or excessive costs, effectively denying service to themselves or others.

Listing 16: LiquidityHub.sol

```
207 function lockNFTCollateral(
```

```
208         CollateralInput[] calldata inputs
209     ) external nonReentrant whenNotPaused {
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Limit the number of NFTs processable per transaction or implement batch processing to manage gas consumption and ensure usability.

Status - Fixed

D RealEstateNFT.sol

D.1 Unrestricted Pause State Modification [LOW]

Description:

The setPaused function allows the factory to toggle the paused state without additional checks. If the factory contract is compromised or misconfigured (e.g., its admin rights are abused), an attacker could pause minting indefinitely,disrupting the NFT collection's functionality and preventing users from minting new tokens. This could harm the project's revenue and user trust.

Listing 17: RealEstateNFT.sol

```
83  function setPaused(bool _paused) external onlyFactory {
84          paused = _paused;
85  }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Introduce a secondary authorization mechanism, such as requiring the super admin's approval alongside the factory, to toggle the pause state, reducing the risk of unauthorized or malicious pausing.

Status - Fixed

D.2 Missing Event for Pause State Change [LOW]

Description:

The setPaused function updates the paused state but does not emit an event. This reduces transparency,making it harder for off-chain systems or users to track when minting is paused or resumed, which could lead to confusion or missed opportunities for minting.

Listing 18: RealEstateNFT.sol

```
83 function setPaused(bool _paused) external onlyFactory {
84         paused = _paused;
85 }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Add an event emission in setPaused to log changes to the pause state, enhancing visibility and enabling better monitoring of contract status.

Status - Fixed

E RealEstateNFTFactory.sol

E.1 Missing Value Checks for pricePerNFT and maxSupply in createCollection [MEDIUM]

Description:

The createCollection function accepts pricePerNFT and maxSupply without validating their values. A pricePerNFT of 0 could result in free NFTs, undermining the economic model, while an excessively high value could make minting impractical. Similarly, a maxSupply of 0 would create a useless collection, and an unbounded high value could lead to unintended inflation or gas issues in downstream contracts like RealEstateNFT. This lack of bounds checking risks creating dysfunctional or exploitable collections that affect the protocol's integrity and usability.

Listing 19: RealEstateNFTFactory.sol

```
40  function createCollection(
41      string memory name,
42      string memory symbol,
43      uint256 pricePerNFT,
44      uint256 maxSupply,
45      string memory projectURI
46  ) external onlyAdmin returns (address) {
47      RealEstateNFT collection = new RealEstateNFT(
48          name,
49          symbol,
50          usdtToken,
51          pricePerNFT,
52          maxSupply,
53          projectURI
54      );

56      collections.push(collection);
```

```
57        isValidCollection[collection] = true;

59        emit CollectionCreated(
60            address(collection),
61            name,
62            symbol,
63            pricePerNFT,
64            maxSupply,
65            projectURI
66        );

68        return address(collection);
69    }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Add input validation to ensure pricePerNFT and maxSupply are within reasonable ranges (e.g., greater than zero and below a sensible upper limit) to enforce a consistent economic model and prevent the creation of invalid or impractical collections.

Status - Fixed

E.2 Gas Inefficiency in setPaused Loop [MEDIUM]

Description:

The setPaused function loops through all collections to update their pause states, which becomes increasingly gas-intensive as the number of collections grows. In a large ecosystem, this could exceed block gas limits, rendering the function unusable and leaving some collections in an inconsistent state (paused or unpaused).

Listing 20: RealEstateNFTFactory.sol

```
71  function setPaused(bool _paused) external onlySuperAdmin {
72      paused = _paused;
73      emit MintingPaused(_paused);

75      // Update pause state for all collections
76      for (uint i = 0; i < collections.length; i++) {
77          collections[i].setPaused(_paused);
78      }
79  }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Implement a batch processing mechanism or delegate pause updates to individual collection admins to distribute gas costs and ensure scalability as the number of collections increases.

Status - Fixed

E.3 No Collection Removal or Invalidation Mechanism [LOW]

Description:

Once a RealEstateNFT collection is created and added to isValidCollection, there's no way to remove or invalidate it (e.g., if a collection is compromised or deprecated). This could allow outdated or insecure collections to remain valid indefinitely, posing a risk to downstream contracts like LiquidityHub that rely on isValidCollection for trust.

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Add a function to invalidate or remove collections from isValidCollection and collections, restricted to the super admin, to maintain control over the set of trusted collections.

Status - Fixed

E.4 Missing Validation of Constructor Inputs [LOW]

Description:

The constructor accepts _adminRegistry and _usdtToken without verifying they are non-zero addresses or valid contracts. If either is mistakenly set to address(0) or an invalid contract, the factory becomes unusable (e.g., adminRegistry.owner() reverts), locking out admin functions and breaking collection creation.

Listing 21: RealEstateNFTFactory.sol

```
35 constructor(address _adminRegistry, address _usdtToken) {
36     adminRegistry = AdminRegistry(_adminRegistry);
37     usdtToken = _usdtToken;
38 }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Add checks in the constructor to ensure _adminRegistry and _usdtToken are non-zero addresses and, where possible, validate their expected interfaces to prevent deployment with

invalid parameters.

Status - Fixed

F AdminRegistry.sol

F.1 No Maximum Admin Limit [LOW]

Description:

The contract allows an unlimited number of admins to be added to the adminList array. This could lead to excessive gas consumption for operations like removeAdmin or getAdminList as the array grows, potentially making the contract inefficient or unusable over time.

Listing 22: AdminRegistry.sol

```
25 function _addAdmin(address admin) internal {
26     require(admin != address(0), "Invalid address");
27     require(!isAdmin[admin], "Already an admin");
28     isAdmin[admin] = true;
29     adminList.push(admin);
30     emit AdminAdded(admin);
31 }
```

Risk Level:

Likelihood – 2
Impact - 1

Recommendation:

Implement a configurable maximum admin limit (e.g., via a state variable and a check in _addAdmin) to prevent unbounded growth of the adminList array. This improves gas efficiency and ensures long-term usability.

Status - Fixed

F.2 Single-Step Ownership Transfer [LOW]

Description:

The contract inherits from OpenZeppelin's Ownable, which uses a single-step ownership transfer. If the super admin's private key is compromised, an attacker could immediately transfer ownership, taking full control of the contract without a chance for recovery.

Listing 23: AdminRegistry.sol

```
5 import "@openzeppelin/contracts/access/Ownable.sol";
6 contract AdminRegistry is Ownable {
7     ...
8 }
```

Risk Level:

Likelihood – 2
Impact - 1

Recommendation:

Replace Ownable with Ownable2Step from OpenZeppelin to implement a two-step ownership transfer process, requiring the new owner to accept the transfer. This adds a layer of security against key compromise.

Status - Fixed

F.3 Gas Inefficiency in Array Removal [LOW]

Description:

The removeAdmin function uses an unordered array removal pattern by swapping the target admin with the last element and popping the array. While gas-efficient, this alters the order of adminList, which could confuse applications expecting a stable order.

Listing 24: AdminRegistry.sol

```
36  for (uint i = 0; i < adminList.length; i++) {
37      if (adminList[i] == admin) {
38          adminList[i] = adminList[adminList.length - 1];
39          adminList.pop();
40          break;
41      }
42  }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Document that adminList order is not preserved after removal, or implement an ordered removal pattern (e.g., shifting elements) if order stability is required, though this increases gas costs.

Status - Fixed

F.4 Floating Pragma Version [LOW]

Description:

The smart contract uses a floating Solidity pragma (0.8.20). This means the compiler may use any minor version within the 0.8.x range, potentially leading to unexpected behavior if a newer minor version introduces changes or deprecations. Using a fixed Solidity version enhances security by ensuring consistency in compilation.

Listing 25: AdminRegistry.sol

```
2  pragma solidity ^0.8.20;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Specify a fixed Solidity version (e.g., pragma solidity 0.8.20;) to ensure consistent compilation and avoid potential issues from unintended compiler updates. Always verify that the chosen version is the most suitable for security and functionality.

Status - Acknowledged

4 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies.Slither, a Solidity static analysis framework,was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats.This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) has bitwise-xor operator ^ instead of the
    ,→ exponentiation operator **:
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#203)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #incorrect-exponentiation
INFO:Detectors:
MRTCollection._tokenURIs (src/MRTCollection.sol#40-41) shadows:
        - ERC721URIStorage._tokenURIs (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/token/
            ,→ ERC721/extensions/ERC721URIStorage.sol#21-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #state-variable-shadowing
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
```

```
        - denominator = denominator / twos (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#185)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#203)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#185)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-
            ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts
            ,→ /utils/math/Math.sol#207)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#185)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-
            ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts
            ,→ /utils/math/Math.sol#208)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#185)
```

```
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-
            ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts
            ,→ /utils/math/Math.sol#209)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#185)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-
            ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts
            ,→ /utils/math/Math.sol#210)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#185)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-
            ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts
            ,→ /utils/math/Math.sol#211)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) performs a multiplication on the result of a
    ,→ division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/
            ,→ lib/openzeppelin-contracts/contracts/utils/math/Math.sol
            ,→ #188-189)
        - result = prod0 * inverse (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#217)
```

```
Math.invMod(uint256,uint256) (lib/openzeppelin-contracts-upgradeable/lib
    ,→ /openzeppelin-contracts/contracts/utils/math/Math.sol#238-281)
    ,→ performs a multiplication on the result of a division:
        - quotient = gcd / remainder (lib/openzeppelin-contracts-
            ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/
            ,→ math/Math.sol#256-258)
        - (gcd,remainder) = (remainder,gcd - remainder * quotient) (lib/
            ,→ openzeppelin-contracts-upgradeable/lib/openzeppelin-
            ,→ contracts/contracts/utils/math/Math.sol#258-270)
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) performs a
    ,→ multiplication on the result of a division:
        - tokenBalance = mrtToken.balanceOf(voter) / 10 ** 18 (src/MRTDAO
            ,→ .sol#180-182)
        - (tokenBalance * totalMultiplier) / 10000 (src/MRTDAO.sol
            ,→ #212-213)
MRTStaking.calculateRewards(uint256,uint256) (src/MRTStaking.sol
    ,→ #136-172) performs a multiplication on the result of a division:
        - daysStaked = claimDuration / 86400 (src/MRTStaking.sol#166-168)
        - reward = rewardRate * daysStaked * multiplier / 10000 (src/
            ,→ MRTStaking.sol#168-172)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #divide-before-multiply
INFO:Detectors:
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) uses a
    ,→ dangerous strict equality:
        - tokenBalance == 0 (src/MRTDAO.sol#183-184)
MRTStaking.calculateRewards(uint256,uint256) (src/MRTStaking.sol
    ,→ #136-172) uses a dangerous strict equality:
        - claimDuration == 0 (src/MRTStaking.sol#151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MRTStaking.emergencyWithdraw(uint256[]) (src/MRTStaking.
    ,→ sol#289-306):
```

```
        External calls:
        - nftCollection.safeTransferFrom(address(this),owner,tokenId) (
            ,→ src/MRTStaking.sol#302-304)
        State variables written after the call(s):
        - stakingInfo.isStaked = false (src/MRTStaking.sol#301-302)
        MRTStaking.tokenStakingInfo (src/MRTStaking.sol#33-35) can be
            ,→ used in cross function reentrancies:
        - MRTStaking.calculateRewards(uint256,uint256) (src/MRTStaking.
            ,→ sol#136-172)
        - MRTStaking.emergencyWithdraw(uint256[]) (src/MRTStaking.sol
            ,→ #289-306)
        - MRTStaking.getStakingInfo(uint256) (src/MRTStaking.sol#272-284)
        - MRTStaking.tokenStakingInfo (src/MRTStaking.sol#33-35)
Reentrancy in MRTStaking.stakeNFT(uint256) (src/MRTStaking.sol#174-195):
        External calls:
        - nftCollection.safeTransferFrom(msg.sender,address(this),tokenId
            ,→ ) (src/MRTStaking.sol#184-185)
        State variables written after the call(s):
        - tokenStakingInfo[tokenId] = StakingInfo({stakedTimestamp:block.
            ,→ timestamp,lastClaimTimestamp:block.timestamp,owner:msg.
            ,→ sender,isStaked:true}) (src/MRTStaking.sol#185-191)
        MRTStaking.tokenStakingInfo (src/MRTStaking.sol#33-35) can be
            ,→ used in cross function reentrancies:
        - MRTStaking.calculateRewards(uint256,uint256) (src/MRTStaking.
            ,→ sol#136-172)
        - MRTStaking.emergencyWithdraw(uint256[]) (src/MRTStaking.sol
            ,→ #289-306)
        - MRTStaking.getStakingInfo(uint256) (src/MRTStaking.sol#272-284)
        - MRTStaking.tokenStakingInfo (src/MRTStaking.sol#33-35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #reentrancy-vulnerabilities-1
INFO:Detectors:
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) ignores
    ,→ return value by (None,None,owner,isStaked) = stakingContract.
```

```
    ,→ getStakingInfo(tokenId_scope_1) (src/MRTDAO.sol#204-206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #unused-return
INFO:Detectors:
MRTDAO.calculateVotingPower(address).owner (src/MRTDAO.sol#205) shadows:
        - Ownable.owner() (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/access/Ownable.sol#54-56)
            ,→ (function)
MRTStaking.getStakingInfo(uint256).owner (src/MRTStaking.sol#275)
    ,→ shadows:
        - Ownable.owner() (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/access/Ownable.sol#54-56)
            ,→ (function)
MRTStaking.emergencyWithdraw(uint256[]).owner (src/MRTStaking.sol#301)
    ,→ shadows:
        - Ownable.owner() (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/access/Ownable.sol#54-56)
            ,→ (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #local-variable-shadowing
INFO:Detectors:
MRTCollection.reduceMaxSupply(uint256) (src/MRTCollection.sol#254-266)
    ,→ should emit an event for:
        - maxSupply -= reductionAmount (src/MRTCollection.sol#264-265)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #missing-events-arithmetic
INFO:Detectors:
MRTCollection.constructor(string,string,string,address,address,address,
    ,→ uint96)._trustedOracle (src/MRTCollection.sol#71) lacks a zero-
    ,→ check on :
                - trustedOracle = _trustedOracle (src/MRTCollection.sol
                    ,→ #77-78)
MRTCollection.setContractAddresses(address,address)._presaleContract (
    ,→ src/MRTCollection.sol#117) lacks a zero-check on :
```

```
                    - presaleContract = _presaleContract (src/MRTCollection.
                        ,→ sol#119-120)
MRTCollection.updateTrustedOracle(address)._trustedOracle (src/
    ,→ MRTCollection.sol#136) lacks a zero-check on :
                    - trustedOracle = _trustedOracle (src/MRTCollection.sol
                        ,→ #137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #missing-zero-address-validation
INFO:Detectors:
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) has
    ,→ external calls inside a loop: tokenId = nftCollection.
    ,→ tokenOfOwnerByIndex(voter,i) (src/MRTDAO.sol#193-195)
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) has
    ,→ external calls inside a loop: rarity = nftCollection.tokenRarity(
    ,→ tokenId) (src/MRTDAO.sol#196-197)
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) has
    ,→ external calls inside a loop: (None,None,owner,isStaked) =
    ,→ stakingContract.getStakingInfo(tokenId_scope_1) (src/MRTDAO.sol
    ,→ #204-206)
MRTDAO.calculateVotingPower(address) (src/MRTDAO.sol#177-213) has
    ,→ external calls inside a loop: rarity_scope_2 = nftCollection.
    ,→ tokenRarity(tokenId_scope_1) (src/MRTDAO.sol#208-209)
MRTStaking.emergencyWithdraw(uint256[]) (src/MRTStaking.sol#289-306) has
    ,→ external calls inside a loop: nftCollection.safeTransferFrom(
    ,→ address(this),owner,tokenId) (src/MRTStaking.sol#302-304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ /#calls-inside-a-loop
INFO:Detectors:
Reentrancy in MRTStaking.addRewards(uint256) (src/MRTStaking.sol
    ,→ #121-129):
        External calls:
        - require(bool,string)(mrtToken.transferFrom(msg.sender,address(
            ,→ this),amount),Token transfer failed) (src/MRTStaking.sol
            ,→ #125-128)
```

```
        State variables written after the call(s):
        - rewardsPool = rewardsPool + amount (src/MRTStaking.sol#128-129)
Reentrancy in MRTCollection.mintInternal(address,bytes,bytes32) (src/
    ,→ MRTCollection.sol#169-208):
        External calls:
        - _safeMint(to,tokenId) (src/MRTCollection.sol#188-190)
                - retval = IERC721Receiver(to).onERC721Received(operator,
                    ,→ from,tokenId,data) (lib/openzeppelin-contracts-
                    ,→ upgradeable/lib/openzeppelin-contracts/contracts/
                    ,→ token/ERC721/utils/ERC721Utils.sol#32-45)
                - ERC721Utils.checkOnERC721Received(_msgSender(),address
                    ,→ (0),to,tokenId,data) (lib/openzeppelin-contracts-
                    ,→ upgradeable/lib/openzeppelin-contracts/contracts/
                    ,→ token/ERC721/ERC721.sol#310)
        State variables written after the call(s):
        - _setTokenURI(tokenId,tokenUriForRarity) (src/MRTCollection.sol
            ,→ #204-205)
                - _tokenURIs[tokenId] = _tokenURI (lib/openzeppelin-
                    ,→ contracts-upgradeable/lib/openzeppelin-contracts/
                    ,→ contracts/token/ERC721/extensions/ERC721URIStorage.
                    ,→ sol#57)
        - tokenRarity[tokenId] = rarity (src/MRTCollection.sol#206-207)
Reentrancy in MRTStaking.stakeNFT(uint256) (src/MRTStaking.sol#174-195):
        External calls:
        - nftCollection.safeTransferFrom(msg.sender,address(this),tokenId
            ,→ ) (src/MRTStaking.sol#184-185)
        State variables written after the call(s):
        - userStakedTokens[msg.sender].push(tokenId) (src/MRTStaking.sol
            ,→ #192-193)
Reentrancy in MRTStaking.unstakeNFT(uint256) (src/MRTStaking.sol
    ,→ #199-237):
        External calls:
        - nftCollection.safeTransferFrom(address(this),msg.sender,tokenId
            ,→ ) (src/MRTStaking.sol#224-225)
```

```
        State variables written after the call(s):
        - rewardsPool = rewardsPool - reward (src/MRTStaking.sol#230-231)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in MRTStaking.addRewards(uint256) (src/MRTStaking.sol
    ,→ #121-129):
        External calls:
        - require(bool,string)(mrtToken.transferFrom(msg.sender,address(
            ,→ this),amount),Token transfer failed) (src/MRTStaking.sol
            ,→ #125-128)
        Event emitted after the call(s):
        - RewardsAdded(amount) (src/MRTStaking.sol#129)
Reentrancy in MRTDAO.addTokenFunds(uint256) (src/MRTDAO.sol#166-173):
        External calls:
        - require(bool,string)(mrtToken.transferFrom(msg.sender,address(
            ,→ this),amount),Token transfer failed) (src/MRTDAO.sol
            ,→ #170-171)
        Event emitted after the call(s):
        - FundsAdded(amount) (src/MRTDAO.sol#171-173)
Reentrancy in MRTStaking.emergencyWithdraw(uint256[]) (src/MRTStaking.
    ,→ sol#289-306):
        External calls:
        - nftCollection.safeTransferFrom(address(this),owner,tokenId) (
            ,→ src/MRTStaking.sol#302-304)
        Event emitted after the call(s):
        - StakingEnded(owner,tokenId,block.timestamp) (src/MRTStaking.sol
            ,→ #304-305)
Reentrancy in MRTCollection.mintInternal(address,bytes,bytes32) (src/
    ,→ MRTCollection.sol#169-208):
        External calls:
        - _safeMint(to,tokenId) (src/MRTCollection.sol#188-190)
                - retval = IERC721Receiver(to).onERC721Received(operator,
                    ,→ from,tokenId,data) (lib/openzeppelin-contracts-
```

```
                    ,→ upgradeable/lib/openzeppelin-contracts/contracts/
                    ,→ token/ERC721/utils/ERC721Utils.sol#32-45)
                - ERC721Utils.checkOnERC721Received(_msgSender(),address
                    ,→ (0),to,tokenId,data) (lib/openzeppelin-contracts-
                    ,→ upgradeable/lib/openzeppelin-contracts/contracts/
                    ,→ token/ERC721/ERC721.sol#310)
        Event emitted after the call(s):
        - MetadataUpdate(tokenId) (lib/openzeppelin-contracts-upgradeable
            ,→ /lib/openzeppelin-contracts/contracts/token/ERC721/
            ,→ extensions/ERC721URIStorage.sol#57-58)
                - _setTokenURI(tokenId,tokenUriForRarity) (src/
                    ,→ MRTCollection.sol#204-205)
        - Minted(to,tokenId,rarity) (src/MRTCollection.sol#207-208)
        - RaritySet(tokenId,rarity) (src/MRTCollection.sol#207)
Reentrancy in MRTDAO.reduceMaxSupply(uint256) (src/MRTDAO.sol#342-352):
        External calls:
        - success = nftCollection.reduceMaxSupply(reductionAmount) (src/
            ,→ MRTDAO.sol#345-349)
        Event emitted after the call(s):
        - MaxSupplyReduced(reductionAmount) (src/MRTDAO.sol#350)
Reentrancy in MRTDAO.withdrawFunds(address,uint256) (src/MRTDAO.sol
    ,→ #315-326):
        External calls:
        - (success,None) = recipient.call{value: amount}() (src/MRTDAO.
            ,→ sol#322-323)
        Event emitted after the call(s):
        - FundsWithdrawn(recipient,amount) (src/MRTDAO.sol#325-326)
Reentrancy in MRTDAO.withdrawTokens(address,uint256) (src/MRTDAO.sol
    ,→ #331-338):
        External calls:
        - require(bool,string)(mrtToken.transfer(recipient,amount),Token
            ,→ transfer failed) (src/MRTDAO.sol#333-335)
        Event emitted after the call(s):
        - FundsWithdrawn(recipient,amount) (src/MRTDAO.sol#338)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #reentrancy-vulnerabilities-3
INFO:Detectors:
MRTDAO.castVote(uint256,bool) (src/MRTDAO.sol#263-285) uses timestamp
    ,→ for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= proposal.startTime,
            ,→ Voting has not started) (src/MRTDAO.sol#269-270)
        - require(bool,string)(block.timestamp <= proposal.endTime,Voting
            ,→ has ended) (src/MRTDAO.sol#270-272)
MRTDAO.executeProposal(uint256) (src/MRTDAO.sol#290-311) uses timestamp
    ,→ for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > proposal.endTime,Voting
            ,→ still in progress) (src/MRTDAO.sol#297-298)
        - require(bool,string)(block.timestamp <= proposal.endTime +
            ,→ executionDelay,Execution window passed) (src/MRTDAO.sol
            ,→ #299-301)
MRTStaking.calculateRewards(uint256,uint256) (src/MRTStaking.sol
    ,→ #136-172) uses timestamp for comparisons
        Dangerous comparisons:
        - claimDuration == 0 (src/MRTStaking.sol#151)
        - stakingDuration >= stakingPeriods[i].days_ * 86400 (src/
            ,→ MRTStaking.sol#162-163)
MRTStaking.stakeNFT(uint256) (src/MRTStaking.sol#174-195) uses timestamp
    ,→ for comparisons
        Dangerous comparisons:
        - require(bool,string)(! tokenStakingInfo[tokenId].isStaked,Token
            ,→ already staked) (src/MRTStaking.sol#181-183)
MRTStaking.unstakeNFT(uint256) (src/MRTStaking.sol#199-237) uses
    ,→ timestamp for comparisons
        Dangerous comparisons:
        - reward > 0 && reward <= rewardsPool (src/MRTStaking.sol
            ,→ #227-228)
```

```
        - require(bool,string)(mrtToken.transfer(msg.sender,reward),
            ,→ Reward transfer failed) (src/MRTStaking.sol#232)
MRTStaking.claimRewards(uint256) (src/MRTStaking.sol#238-260) uses
    ,→ timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stakingInfo.isStaked,Token not staked) (
            ,→ src/MRTStaking.sol#245-246)
        - require(bool,string)(stakingInfo.owner == msg.sender,Not
            ,→ staking owner) (src/MRTStaking.sol#246-248)
        - require(bool,string)(reward > 0,No rewards to claim) (src/
            ,→ MRTStaking.sol#250-251)
        - require(bool,string)(reward <= rewardsPool,Insufficient rewards
            ,→ in pool) (src/MRTStaking.sol#251-252)
        - require(bool,string)(mrtToken.transfer(msg.sender,reward),
            ,→ Reward transfer failed) (src/MRTStaking.sol#257-259)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #block-timestamp
INFO:Detectors:
ERC721Utils.checkOnERC721Received(address,address,address,uint256,bytes)
    ,→ (lib/openzeppelin-contracts-upgradeable/lib/openzeppelin-
    ,→ contracts/contracts/token/ERC721/utils/ERC721Utils.sol#23-46)
    ,→ uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/token/ERC721/utils/
            ,→ ERC721Utils.sol#42-44)
Panic.panic(uint256) (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/Panic.sol#49-53) uses
    ,→ assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/Panic.sol#50-53)
Strings.toString(uint256) (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/Strings.sol#35-52) uses
    ,→ assembly
```

```
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/Strings.sol#40-43)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/Strings.sol#45-48)
Strings.toChecksumHexString(address) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/Strings.
    ,→ sol#100-116) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/Strings.sol
            ,→ #106-108)
Strings._unsafeReadBytesOffset(bytes,uint256) (lib/openzeppelin-
    ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts/utils/
    ,→ Strings.sol#421-432) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/Strings.sol
            ,→ #430-432)
ECDSA.tryRecover(bytes32,bytes) (lib/openzeppelin-contracts-upgradeable/
    ,→ lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol
    ,→ #54-73) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/cryptography/ECDSA.
            ,→ sol#64-69)
MessageHashUtils.toEthSignedMessageHash(bytes32) (lib/openzeppelin-
    ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts/utils/
    ,→ cryptography/MessageHashUtils.sol#28-34) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/cryptography/
            ,→ MessageHashUtils.sol#30-34)
MessageHashUtils.toTypedDataHash(bytes32,bytes32) (lib/openzeppelin-
    ,→ contracts-upgradeable/lib/openzeppelin-contracts/contracts/utils/
    ,→ cryptography/MessageHashUtils.sol#71-80) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/cryptography/
            ,→ MessageHashUtils.sol#75-80)
```

```
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#141-218) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/math/Math.sol
            ,→ #149-150)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/math/Math.sol
            ,→ #171-179)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/math/Math.sol
            ,→ #184-193)
Math.tryModExp(uint256,uint256,uint256) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#332-352) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/math/Math.sol
            ,→ #334-352)
Math.tryModExp(bytes,bytes,bytes) (lib/openzeppelin-contracts-
    ,→ upgradeable/lib/openzeppelin-contracts/contracts/utils/math/Math.
    ,→ sol#366-391) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/math/Math.sol
            ,→ #378-391)
SafeCast.toUint(bool) (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/math/SafeCast.sol
    ,→ #1120-1124) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/utils/math/SafeCast.sol
            ,→ #1122-1123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #assembly-usage
INFO:Detectors:
```

```
MRTCollection._exists(uint256) (src/MRTCollection.sol#276-279) is never
    ,→ used and should be removed
MRTCollection._increaseBalance(address,uint128) (src/MRTCollection.sol
    ,→ #304-308) is never used and should be removed
MRTDAO.reduceMaxSupply(uint256) (src/MRTDAO.sol#342-352) is never used
    ,→ and should be removed
MRTDAO.withdrawFunds(address,uint256) (src/MRTDAO.sol#315-326) is never
    ,→ used and should be removed
MRTDAO.withdrawTokens(address,uint256) (src/MRTDAO.sol#331-338) is never
    ,→ used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #dead-code
INFO:Detectors:
Version constraint ^0.8.20 contains known severe issues (https://
    ,→ solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess.
It is used by:
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/access/Ownable.sol#2-4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/interfaces/IERC165.sol
            ,→ #2-4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/interfaces/IERC2981.sol
            ,→ #2-4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/interfaces/IERC4906.sol
            ,→ #2-4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
            ,→ openzeppelin-contracts/contracts/interfaces/IERC721.sol
            ,→ #2-4)
```

```
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/interfaces/draft-IERC6093
    ,→ .sol#2-3)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC20/IERC20.sol
    ,→ #2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/ERC721.sol
    ,→ #2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/IERC721.sol
    ,→ #2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/
    ,→ IERC721Receiver.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/extensions/
    ,→ ERC721Enumerable.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/extensions/
    ,→ ERC721URIStorage.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/extensions/
    ,→ IERC721Enumerable.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/extensions/
    ,→ IERC721Metadata.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/utils/
    ,→ ERC721Holder.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/ERC721/utils/
    ,→ ERC721Utils.sol#2-4)
```

```
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/token/common/ERC2981.sol
    ,→ #2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/Context.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/Panic.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/ReentrancyGuard.sol
    ,→ #2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/Strings.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/cryptography/ECDSA.
    ,→ sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/cryptography/
    ,→ MessageHashUtils.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/introspection/
    ,→ ERC165.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/introspection/
    ,→ IERC165.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/math/Math.sol#2-4)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/math/SafeCast.sol
    ,→ #3-5)
- ^0.8.20 (lib/openzeppelin-contracts-upgradeable/lib/
    ,→ openzeppelin-contracts/contracts/utils/math/SignedMath.sol
    ,→ #2-4)
- ^0.8.20 (src/MRTCollection.sol#1-2)
- ^0.8.20 (src/MRTDAO.sol#1-2)
```

```
        - ^0.8.20 (src/MRTStaking.sol#1-2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #incorrect-versions-of-solidity
INFO:Detectors:
Low level call in MRTDAO.executeProposal(uint256) (src/MRTDAO.sol
    ,→ #290-311):
        - (success,None) = proposal.targetContract.call(proposal.callData
            ,→ ) (src/MRTDAO.sol#306-307)
Low level call in MRTDAO.withdrawFunds(address,uint256) (src/MRTDAO.sol
    ,→ #315-326):
        - (success,None) = recipient.call{value: amount}() (src/MRTDAO.
            ,→ sol#322-323)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #low-level-calls
INFO:Detectors:
MRTStaking (src/MRTStaking.sol#17-306) should inherit from IMRTStaking (
    ,→ src/MRTDAO.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #missing-inheritance
INFO:Detectors:
Parameter MRTCollection.setContractAddresses(address,address).
    ,→ _presaleContract (src/MRTCollection.sol#117) is not in mixedCase
Parameter MRTCollection.setContractAddresses(address,address).
    ,→ _daoContract (src/MRTCollection.sol#117) is not in mixedCase
Parameter MRTCollection.updateTrustedOracle(address)._trustedOracle (src
    ,→ /MRTCollection.sol#136) is not in mixedCase
Parameter MRTCollection.setRarityURI(MRTCollection.Rarity,string).
    ,→ _tokenURI (src/MRTCollection.sol#144-145) is not in mixedCase
Parameter MRTCollection.setDefaultRoyalty(uint96)._royaltyPercentage (
    ,→ src/MRTCollection.sol#154) is not in mixedCase
Parameter MRTDAO.updateStakingContract(address)._stakingContractAddress
    ,→ (src/MRTDAO.sol#119-120) is not in mixedCase
Parameter MRTDAO.updateVotingParameters(uint256,uint256,uint256).
    ,→ _votingPeriod (src/MRTDAO.sol#130-131) is not in mixedCase
```

```
Parameter MRTDAO.updateVotingParameters(uint256,uint256,uint256).
    ,→ _executionDelay (src/MRTDAO.sol#131) is not in mixedCase
Parameter MRTDAO.updateVotingParameters(uint256,uint256,uint256).
    ,→ _minProposalThreshold (src/MRTDAO.sol#131-133) is not in
    ,→ mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #conformance-to-solidity-naming-conventions
INFO:Detectors:
MRTDAO.constructor(address,address,address) (src/MRTDAO.sol#98-114) uses
    ,→ literals with too many digits:
        - nftVotingPowerMultiplier[IMRTCollection.Rarity.EPIC] = 100000 (
            ,→ src/MRTDAO.sol#112-113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #too-many-digits
INFO:Detectors:
MRTCollection._tokenURIs (src/MRTCollection.sol#40-41) is never used in
    ,→ MRTCollection (src/MRTCollection.sol#14-308)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ,→ #unused-state-variable
INFO:Detectors:
Loop condition i < stakingPeriods.length (src/MRTStaking.sol#161-162)
    ,→ should use cached array length instead of referencing `length`
    ,→ member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-
     ,→ Documentation#cache-array-length
INFO:Detectors:
MRTCollection.mrtToken (src/MRTCollection.sol#26-27) should be immutable
MRTDAO.mrtToken (src/MRTDAO.sol#34-36) should be immutable
MRTDAO.nftCollection (src/MRTDAO.sol#33-34) should be immutable
MRTStaking.mrtToken (src/MRTStaking.sol#24-26) should be immutable
MRTStaking.nftCollection (src/MRTStaking.sol#21-22) should be immutable
MRTStaking.nftCollectionRarity (src/MRTStaking.sol#22-23) should be
    ,→ immutable
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
,→ #state-variables-that-could-be-declared-immutable
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

In this audit, we examined the design and implementation of Principal Protocol MVP contract and discovered several issues of varying severity. Principal Protocol team addressed issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Nexarias auditors advised Principal Protocol Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

For a Smart Contract Audit, contact us at contact@nexarias.com