F Y E O

Security Code Review of RAAC

RAAC

October 2024
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

Executive Summary

Overview

RAAC engaged FYEO Inc. to perform a Security Code Review of RAAC.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on October 01 - October 07, 2024, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-RAAC-01 – Buyback overwrites highest bidder
- FYEO-RAAC-02 – Lock function overwrites data
- FYEO-RAAC-03 – Outdated prices are not handled
- FYEO-RAAC-04 – Dust transfer function allows transfer of any amount
- FYEO-RAAC-05 – Use the safeTransfer function
- FYEO-RAAC-06 – Bid can be increased by tiny amounts
- FYEO-RAAC-07 – LPToken has minter but mint is onlyOwner
- FYEO-RAAC-08 – Missing bound checks & events
- FYEO-RAAC-09 – Repay on behalf of uses wrong debt
- FYEO-RAAC-10 – Users can overpay

- FYEO-RAAC-11 – Assumption of 18 decimals
- FYEO-RAAC-12 – Code does not follow Checks-Effects-Interactions pattern
- FYEO-RAAC-13 – Governance contract assumptions
- FYEO-RAAC-14 – Inconsistent use of updateStateBefore modifier
- FYEO-RAAC-15 – Missing zero checks
- FYEO-RAAC-16 – Tokens remain in auction contract if not sold

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of RAAC. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/RegnumAurumAcquisitionCorp/core with the commit hash 5ffdcd7440fbbdb5a28f14f66987cdbe7fa8b923.

Remediations were provided with the commit hash 4d1bbe8b846fd57492ecd310eaf302a9d5426aa3.

Files included in the code review

```
raac-core/
└── contracts/
    ├── core/
    │   ├── collectors/
    │   │   └── FeeCollector/
    │   │       └── FeeCollector.sol
    │   ├── governance/
    │   │   ├── GaugeController.sol
    │   │   └── Governance.sol
    │   ├── minters/
    │   │   └── RAACMinter/
    │   │       └── RAACMinter.sol
    │   ├── pools/
    │   │   ├── LendingPool/
    │   │   │   └── LendingPool.sol
    │   │   └── StabilityPool/
    │   │       ├── MarketCreator.sol
    │   │       ├── NFTLiquidator.sol
    │   │       └── StabilityPool.sol
    │   │
```

Files included in the code review

```
    │   ├── primitives/
    │   │   └── RAACHousePrices.sol
    │   └── tokens/
    │       ├── DEToken.sol
    │       ├── DebtToken.sol
    │       ├── IndexToken.sol
    │       ├── LPToken.sol
    │       ├── RAACNFT.sol
    │       ├── RAACToken.sol
    │       ├── RToken.sol
    │       └── veRAACToken.sol
    ├── interfaces/
    │   ├── IDEToken.sol
    │   ├── IDebtToken.sol
    │   ├── IERC20.sol
    │   ├── IFeeCollector.sol
    │   ├── IGaugeController.sol
    │   ├── IGovernance.sol
    │   ├── IHousePrices.sol
    │   ├── ILendingPool.sol
    │   ├── ILiquidityPool.sol
    │   ├── IMarketCreator.sol
    │   ├── INFTLiquidator.sol
    │   ├── IPoolManager.sol
    │   ├── IRAACHousePrices.sol
    │   ├── IRAACMinter.sol
    │   ├── IRAACNFT.sol
    │   ├── IRAACToken.sol
    │   ├── IRToken.sol
    │   ├── IReserveAllocationLibraryMock.sol
    │   ├── IStabilityPool.sol
    │   ├── IUSDC.sol
    │   ├── IveRAACDistributor.sol
    │   └── IveRAACToken.sol
    ├── libraries/
    │   ├── math/
    │   │   ├── PercentageMath.sol
    │   │   └── WadRayMath.sol
    │   ├── DataTypes.sol
    │   └── ReserveLibrary.sol
    └── zeno/
        ├── Auction.sol
        ├── AuctionFactory.sol
        ├── ZENO.sol
        └── ZENOFactory.sol
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of RAAC, we discovered:

- 3 findings with HIGH severity rating.
- 2 findings with MEDIUM severity rating.
- 5 findings with LOW severity rating.
- 6 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

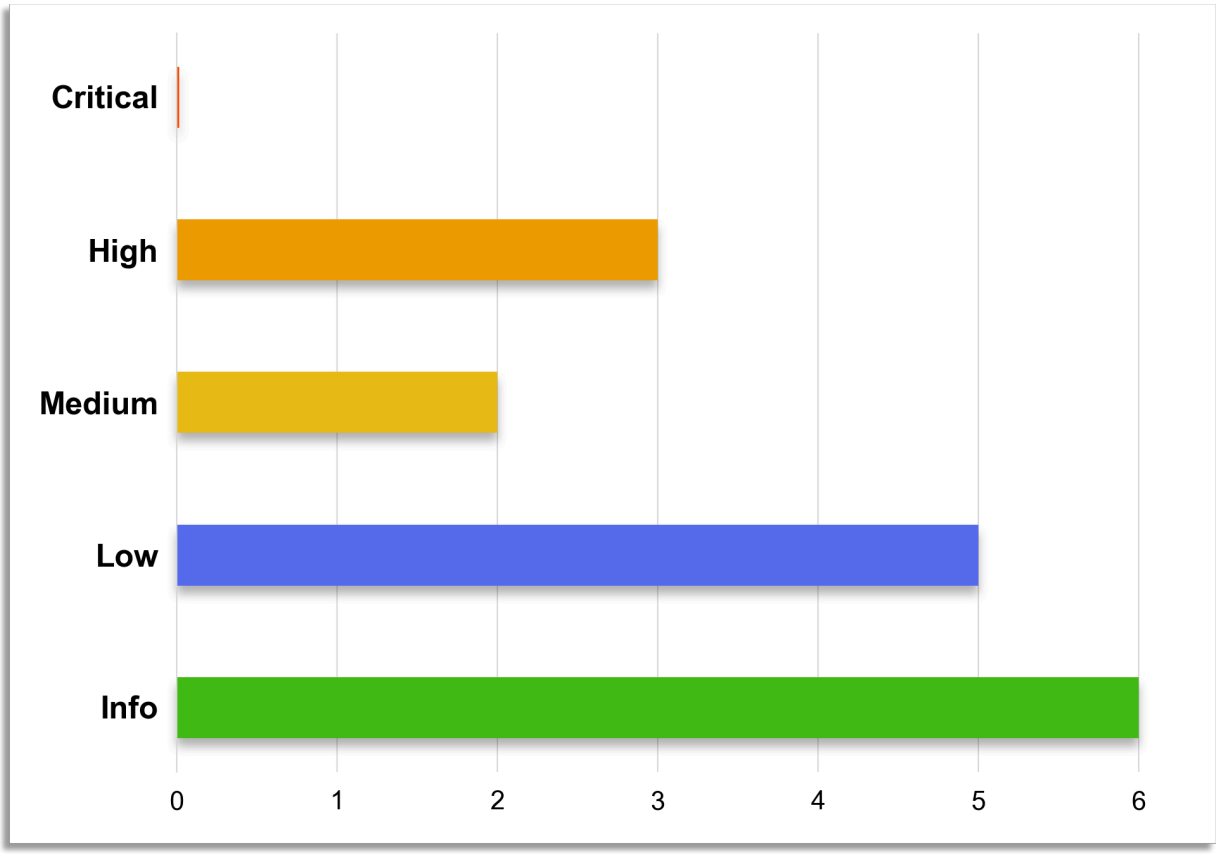

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-RAAC-01	**High**	Buyback overwrites highest bidder
FYEO-RAAC-02	**High**	Lock function overwrites data
FYEO-RAAC-03	**High**	Outdated prices are not handled
FYEO-RAAC-04	**Medium**	Dust transfer function allows transfer of any amount
FYEO-RAAC-05	**Medium**	Use the safeTransfer function
FYEO-RAAC-06	**Low**	Bid can be increased by tiny amounts
FYEO-RAAC-07	**Low**	LPToken has minter but mint is onlyOwner
FYEO-RAAC-08	**Low**	Missing bound checks & events
FYEO-RAAC-09	**Low**	Repay on behalf of uses wrong debt
FYEO-RAAC-10	**Low**	Users can overpay
FYEO-RAAC-11	**Informational**	Assumption of 18 decimals
FYEO-RAAC-12	**Informational**	Code does not follow Checks-Effects-Interactions pattern
FYEO-RAAC-13	**Informational**	Governance contract assumptions
FYEO-RAAC-14	**Informational**	Inconsistent use of updateStateBefore modifier
FYEO-RAAC-15	**Informational**	Missing zero checks
FYEO-RAAC-16	**Informational**	Tokens remain in auction contract if not sold

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

RAAC is a protocol that brings real estate onto the blockchain to provide stability, composability, and accessibility within Decentralized Finance (DeFi). The Regna Minima NFTs are real estate-backed assets. These NFTs are 1:1 backed by properties held in a Special Purpose Vehicle (SPV), require no management or KYC/AML for purchase, and are fully compliant. This model taps into liquidity from secondary NFT markets, making real estate accessible to the DeFi community.

The RAAC ecosystem is composed of several key components. The lending system allows users to deposit or withdraw real estate-backed NFTs and borrow ERC20 tokens against them, while a stability pool manages deposits of ERC20 tokens and rewards users with additional tokens. There is also a minting system linked to the stability pool, responsible for maintaining consensus and minting tokens. Liquidity pools facilitate the creation of specific markets and manage ERC20 interactions, and a foreclosure mechanism handles the liquidation and auctioning of NFT loans. Each part works together to create a fully functional real estate-backed DeFi ecosystem.

Buyback overwrites highest bidder

Finding ID: FYEO-RAAC-01
Severity: **High**
Status: **Remediated**

Description

The buyback function does not consider if someone has bid on the item. The data will be deleted and any funds transferred for bids will be lost. Also, the ownerOf the token won't be the original owner, as the token is transferred to the contract.

Proof of Issue

File name: contracts/core/pools/StabilityPool/NFTLiquidator.sol
Line number: 134

```
function buyBackNFT(uint256 tokenId) external payable {
    TokenData storage data = tokenData[tokenId];
    require(block.timestamp < data.auctionEndTime, "Auction ended");
    require(msg.sender == nftContract.ownerOf(tokenId), "Not the original owner");
    uint256 price = data.debt * 11 / 10; // 110% of the debt
    require(msg.value >= price, "Insufficient payment");

    delete tokenData[tokenId];
```

Severity and Impact Summary

If someone has bid on this token, they will lose their funds as the data will be cleared.

Recommendation

Make sure to refund the highest bidder if set.

Lock function overwrites data

Finding ID: FYEO-RAAC-02
Severity: **High**
Status: **Remediated**

Description

The lock function overwrites data in `locks` if already present.

Proof of Issue

File name: contracts/core/tokens/veRAACToken.sol
Line number: 47

```
function lock(address account, uint256 amount, uint256 lockDuration) external override
nonReentrant {
    require(amount > 0, "Amount must be greater than 0");
    require(lockDuration >= MIN_LOCK_DURATION && lockDuration <= MAX_LOCK_DURATION,
"Invalid lock duration");

    if (msg.sender != minter) {
        require(account == msg.sender, "Can only lock for self");
        raacToken.safeTransferFrom(msg.sender, address(this), amount);
    } else {
        require(account != address(0), "Invalid account");
    }

    uint256 veAmount = calculateVeAmount(amount, lockDuration);
    _mint(account, veAmount);

    locks[account] = Lock(amount, block.timestamp + lockDuration);

    emit Locked(account, amount, lockDuration);
}
```

Severity and Impact Summary

This will lead to a loss of funds.

Recommendation

Make sure to check if this data is already set. Users should use the extend function instead.

Outdated prices are not handled

Finding ID: FYEO-RAAC-03
Severity: **High**
Status: **Remediated**

Description

The oracle uses a global timestamp for when any price was last updated. This time is then returned by the `getLatestPrice` function as if it was actually updated at that time. Which is most likely not the case.

Proof of Issue

File name: contracts/core/primitives/RAACHousePrices.sol
Line number: 76

```
function setHousePrice(uint256 _tokenId, uint256 _amount) external onlyOwner {
    require(_amount > 0, "Price must be greater than 0");
    tokenToHousePrice[_tokenId] = _amount;
    lastUpdateTimestamp = block.timestamp;
    emit PriceUpdated(_tokenId, _amount);
}

function getLatestPrice(uint256 _tokenId) external view returns (uint256, uint256) {
    return (tokenToHousePrice[_tokenId], lastUpdateTimestamp);
}
```

File name: contracts/core/pools/LendingPool/LendingPool.sol
Line number: 600

```
function getNFTPrice(uint256 tokenId) public view returns (uint256) {
    (uint256 price, ) = priceOracle.getLatestPrice(tokenId);
    if (price == 0) revert InvalidNFTPrice();
    return price;
}
```

This function ignores the time the price was last updated.

Severity and Impact Summary

The code does not care about when prices have been updated, if ever. This could lead to a loss of funds.

Recommendation

Make sure to use up to date price info.

Dust transfer function allows transfer of any amount

Finding ID: FYEO-RAAC-04
Severity: **Medium**
Status: **Remediated**

Description

The function is supposed to allow the transfer of dust amounts but allows any amount to be transferred.

Proof of Issue

File name: contracts/core/tokens/RToken.sol
Line number: 401

```
function transferAccruedDust(address recipient, uint256 amount) external onlyReservePool
{
    require(recipient != address(0), "RToken: Recipient cannot be zero address");

    // Calculate the actual balance of the underlying asset held by this contract
    uint256 contractBalance =
IERC20(_assetAddress).balanceOf(address(this)).rayDiv(ILendingPool(_reservePool).getNorm
alizedIncome());

    // Calculate the total real obligations to the token holders
    uint256 totalSupply = totalSupply();
    uint256 totalRealBalance =
totalSupply.rayMul(ILendingPool(_reservePool).getNormalizedIncome());
    // Check if there is any dust (contractBalance > totalRealBalance)
    if (contractBalance > totalRealBalance) {
        uint256 dust = contractBalance - totalRealBalance;
        if(amount > dust){
            dust = amount;
        }
        IERC20(_assetAddress).safeTransfer(recipient, dust);
    }
```

Severity and Impact Summary

If the amount is greater than dust, `amount` will be used as the amount of tokens to transfer. Therefore any requested amount can be transferred out by the reserve pool.

Recommendation

Modify the if statement to reflect the intended functionality.

Use the safeTransfer function

Finding ID: FYEO-RAAC-05
Severity: **Medium**
Status: **Remediated**

Description

The redeem functions do not use `safeTransfer` and burn after transferring.

Proof of Issue

File name: contracts/zeno/ZENO.sol
Line number: 76

```
function redeem(uint amount) external {
    if (!isRedeemable()) {
        revert BondNotRedeemable();
    }

    if (amount == 0) {
        revert ZeroAmount();
    }

    uint256 totalAmount = balanceOf(msg.sender);
    if (amount > totalAmount) {
        revert InsufficientBalance();
    }

    totalZENORedeemed += amount;
    USDC.transfer(msg.sender, amount);
    _burn(msg.sender, amount);
}

function redeemAll() external {
    if (!isRedeemable()) {
        revert BondNotRedeemable();
    }

    uint256 amount = balanceOf(msg.sender);
    totalZENORedeemed += amount;
    USDC.transfer(msg.sender, amount);
    _burn(msg.sender, amount);
}
```

Severity and Impact Summary

The tokens may not be transferred. There is also a risk of reentrancy.

Recommendation

Make sure to burn before transferring or use a reentrancy guard. Also make sure to use the `safeTransfer` function.

Bid can be increased by tiny amounts

Finding ID: FYEO-RAAC-06
Severity: **Low**
Status: **Remediated**

Description

The bid can be increased 1 Wei at a time.

Proof of Issue

File name: contracts/core/pools/StabilityPool/NFTLiquidator.sol
Line number: 89

```
function placeBid(uint256 tokenId) external payable {
    TokenData storage data = tokenData[tokenId];
    require(block.timestamp < data.auctionEndTime, "Auction ended");
    require(msg.value > data.highestBid, "Bid too low");
```

Severity and Impact Summary

This makes outbidding someone fairly cost efficient.

Recommendation

Maybe add a percentage based threshold.

LPToken has minter but mint is onlyOwner

Finding ID: FYEO-RAAC-07
Severity: **Low**
Status: **Remediated**

Description

The LPToken has a `minter` role but the `mint` function is `onlyOwner`.

Proof of Issue

File name: contracts/core/tokens/LPToken.sol
Line number: 14

```
address public minter;

...

function mint(address to, uint256 amount) external onlyOwner {
    _mint(to, amount);
```

Severity and Impact Summary

The code may not be implemented as desired.

Recommendation

Make sure to implement this code according to requirements.

Missing bound checks & events

Finding ID: FYEO-RAAC-08
Severity: **Low**
Status: **Remediated**

Description

Proof of Issue

File name: contracts/core/minters/RAACMinter/RAACMinter.sol
Line number: 308

```
function updateBenchmarkRate(uint256 _newRate) external onlyRole(UPDATER_ROLE) {
    benchmarkRate = _newRate;
    emit ParameterUpdated("benchmarkRate", _newRate);
}

function setMinEmissionRate(uint256 _minEmissionRate) external onlyRole(UPDATER_ROLE) {
    minEmissionRate = _minEmissionRate;
    emit ParameterUpdated("minEmissionRate", _minEmissionRate);
}

function setMaxEmissionRate(uint256 _maxEmissionRate) external onlyRole(UPDATER_ROLE) {
    maxEmissionRate = _maxEmissionRate;
    emit ParameterUpdated("maxEmissionRate", _maxEmissionRate);
}

function setAdjustmentFactor(uint256 _adjustmentFactor) external onlyRole(UPDATER_ROLE)
{
    adjustmentFactor = _adjustmentFactor;
    emit ParameterUpdated("adjustmentFactor", _adjustmentFactor);
}

function setUtilizationTarget(uint256 _utilizationTarget) external
onlyRole(UPDATER_ROLE) {
    utilizationTarget = _utilizationTarget;
    emit ParameterUpdated("utilizationTarget", _utilizationTarget);
}

function setEmissionUpdateInterval(uint256 _emissionUpdateInterval) external
onlyRole(UPDATER_ROLE) {
    emissionUpdateInterval = _emissionUpdateInterval;
    emit ParameterUpdated("emissionUpdateInterval", _emissionUpdateInterval);
}
```

File name: contracts/core/pools/LendingPool/LendingPool.sol
Line number: 644

```
function setPrimeRate(uint256 newPrimeRate) external onlyOwner {
    ReserveLibrary.setPrimeRate(reserve, rateData, newPrimeRate);
} // No event is emitted

function setProtocolFeeRate(uint256 newProtocolFeeRate) external onlyOwner {
```

```
    rateData.protocolFeeRate = newProtocolFeeRate;
} // No bounds, no event

function setLiquidationThreshold(uint256 newLiquidationThreshold) external onlyOwner {
    liquidationThreshold = newLiquidationThreshold;
    emit LiquidationParametersUpdated(liquidationThreshold,
healthFactorLiquidationThreshold, liquidationGracePeriod);
} // No bounds

function setHealthFactorLiquidationThreshold(uint256
newHealthFactorLiquidationThreshold) external onlyOwner {
    healthFactorLiquidationThreshold = newHealthFactorLiquidationThreshold;
    emit LiquidationParametersUpdated(liquidationThreshold,
healthFactorLiquidationThreshold, liquidationGracePeriod);
} // No bounds

function setLiquidationGracePeriod(uint256 newLiquidationGracePeriod) external onlyOwner
{
    liquidationGracePeriod = newLiquidationGracePeriod;
    emit LiquidationParametersUpdated(liquidationThreshold,
healthFactorLiquidationThreshold, liquidationGracePeriod);
} // No bounds
```

File name: contracts/core/pools/StabilityPool/MarketCreator.sol
Line number: 47

```
function createMarket(address _quoteAsset, uint256 _lockDuration, uint256 _reward)
external onlyOwner {
    require(_quoteAsset != address(0), "Invalid quote asset address");
    require(_lockDuration > 0, "Lock duration must be greater than 0");
    marketCount++;
    markets[marketCount] = Market(IERC20(_quoteAsset), _lockDuration, _reward, 0);
```

Check both upper & lower bounds on `_lockDuration` and `_reward`.

File name: contracts/zeno/ZENO.sol
Line number: 24

```
constructor(
    address _usdc,
    uint256 _maturityDate,
    string memory _name,
    string memory _symbol,
    address _initialOwner
) Ownable(_initialOwner) ERC20(_name, _symbol) {
    USDC = IERC20(_usdc);
    MATURITY_DATE = _maturityDate;
```

There are no bounds on `_maturityDate`.

File name: contracts/core/tokens/RAACToken.sol
Line number: 76

```
constructor(
    address initialOwner,
    uint256 initialSwapTaxRate,
    uint256 initialBurnTaxRate
) ERC20("RAAC Token", "RAAC") Ownable(initialOwner) {
```

```
    feeCollector = initialOwner;
    swapTaxRate = initialSwapTaxRate == 0 ? 100 : initialSwapTaxRate; // Default 1% if
not set
    burnTaxRate = initialBurnTaxRate == 0 ? 50 : initialBurnTaxRate; // Default 0.5% if
not set
    if (swapTaxRate > PercentageMath.PERCENTAGE_FACTOR) revert
SwapTaxRateExceedsLimit();
    if (burnTaxRate > PercentageMath.PERCENTAGE_FACTOR) revert
BurnTaxRateExceedsLimit();

...

function setSwapTaxRate(uint256 _swapTaxRate) external onlyOwner {
    if (_swapTaxRate > MAX_TAX_RATE) revert TaxRateExceedsLimit();

    uint256 maxIncrease = swapTaxRate.percentMul(taxRateIncrementLimit);
    uint256 maxDecrease = swapTaxRate.percentMul(taxRateIncrementLimit);
```

There are no lower bounds on `initialSwapTaxRate`, `initialBurnTaxRate` and `_swapTaxRate`. Also `maxIncrease` and `maxDecrease` will be identical.

File name: contracts/zeno/Auction.sol
Line number: 57

```
constructor(
    address _zenoAddress,
    address _usdcAddress,
    address _businessAddress,
    uint256 _auctionStartTime,
    uint256 _auctionEndTime,
    uint256 _startingPrice,
    uint256 _reservePrice,
    uint256 _totalZENOAllocated,
    address _initialOwner
) Ownable(_initialOwner) {
    zeno = ZENO(_zenoAddress);
    usdc = IUSDC(_usdcAddress);
    businessAddress = _businessAddress;
    auctionStartTime = _auctionStartTime;
    auctionEndTime = _auctionEndTime;
    startingPrice = _startingPrice;
    reservePrice = _reservePrice;
    totalZENOAllocated = _totalZENOAllocated;
    totalZENORemaining = _totalZENOAllocated;
}

...
function getPrice() public view returns (uint256) {
    if (block.timestamp >= auctionEndTime) {
        return reservePrice;
    }
    uint256 elapsed = block.timestamp - auctionStartTime;
    uint256 duration = auctionEndTime - auctionStartTime;
```

The start time can be greater than the end time. Neither have a relation to the current time. The end time could be in a billion years. These can cause an underflow & a DoS situation in `getPrice`.

Severity and Impact Summary

Missing bound checks could cause issues by accident or deliberate action.

Recommendation

Make sure to establish some bounds for configuration options that guarantee proper operation of the program.

Repay on behalf of uses wrong debt

Finding ID: FYEO-RAAC-09
Severity: **Low**
Status: **Remediated**

Description

The repay function allows repayments on behalf of other users. It does use the debt of the message sender however.

Proof of Issue

File name: contracts/core/pools/LendingPool/LendingPool.sol
Line number: 425

```
function repay(uint256 amount) external nonReentrant whenNotPaused
onlyValidAmount(amount) {
    address onBehalfOf = msg.sender;
    UserData storage user = userData[onBehalfOf];

    // Update reserve state before repayment
    ReserveLibrary.updateReserveState(reserve, rateData);

    // If amount is greater than userDebt, cap it at userDebt
    uint256 userDebt =
IDebtToken(reserve.reserveDebtTokenAddress).balanceOf(msg.sender);
    uint256 userScaledDebt = userDebt.rayDiv(reserve.usageIndex);
    uint256 actualRepayAmount = amount > userScaledDebt ? userScaledDebt : amount;
```

Severity and Impact Summary

The debt calculated should be from the user the debt is being repaid for.

Recommendation

Use the balance of the correct user.

Users can overpay

Finding ID: FYEO-RAAC-10
Severity: **Low**
Status: **Remediated**

Description

The function checks the amount sent against the price expecting it to be equal or greater. That implies that users can choose to pay extra.

Proof of Issue

File name: contracts/core/tokens/RAACNFT.sol
Line number: 35

```
function mint(uint256 _tokenId, uint256 _amount) public override {
    uint256 price = raac_hp.tokenToHousePrice(_tokenId);
    if(price == 0) { revert RAACNFT__HousePrice(); }
    if(price > _amount) { revert RAACNFT__InsufficientFundsMint(); }

    // transfer erc20 from user to contract - requires pre-approval from user
    token.safeTransferFrom(msg.sender, address(this), _amount);

    // mint tokenId to user
    _safeMint(msg.sender, _tokenId);
}
```

Severity and Impact Summary

Users can send more tokens than required.

Recommendation

Make sure that this functionality is implemented as desired.

Assumption of 18 decimals

Finding ID: FYEO-RAAC-11
Severity: **Informational**
Status: **Remediated**

Description

Functions throughout the code base assume that certain tokens must have 18 decimals. Neither constructors nor setters ever check this to be the case.

Proof of Issue

File name: contracts/core/pools/StabilityPool/StabilityPool.sol
Line number: 217

```
function calculateDeCRVUSDAmount(uint256 rcrvUSDAmount) public view returns (uint256) {
    return (rcrvUSDAmount * 1e18) / getExchangeRate();
}
```

File name: contracts/core/pools/StabilityPool/StabilityPool.sol
Line number: 234

```
function getExchangeRate() public view returns (uint256) {
    // uint256 totalDeCRVUSD = deToken.totalSupply();
    // uint256 totalRcrvUSD = rToken.balanceOf(address(this));
    // if (totalDeCRVUSD == 0 || totalRcrvUSD == 0) return 1e18;

    // uint256 exchangeRate = (totalRcrvUSD * 1e18) / totalDeCRVUSD;
    return 1e18;
    // return exchangeRate;
}
```

Severity and Impact Summary

If the tokens do not have the required number of decimals, calculations will be wrong.

Recommendation

Make sure to enforce checks on decimals when math operations depend on it.

Code does not follow Checks-Effects-Interactions pattern

Finding ID: FYEO-RAAC-12
Severity: **Informational**
Status: **Acknowledged**

Description

The code does not use re-entrancy guards or does not follow the Checks-Effects-Interactions pattern.

Proof of Issue

File name: contracts/zeno/Auction.sol
Line number: 96

```
function buy(uint256 amount) external auctionOngoing {
    require(amount <= totalZENORemaining, "Not enough ZENO remaining");
    uint256 price = getPrice();
    uint256 cost = price * amount;

    require(
        usdc.transferFrom(msg.sender, businessAddress, cost),
        "USDC transfer failed"
    );

    // Create a new bidder
    Bid memory bid = Bid(msg.sender, amount);
    bids.push(bid);
    bidAmounts[msg.sender] += amount;

    lastBidTime = block.timestamp;
    lastBidder = msg.sender;

    // Update the total ZENO remaining
    totalZENORemaining -= amount;
    // Keep track of the total amount of ZENO purchased
    zeno.mint(msg.sender, amount);
    emit ZENOPurchased(msg.sender, amount, price);
}
```

File name: contracts/libraries/ReserveLibrary.sol
Line number: 328

```
function deposit(ReserveData storage reserve,ReserveRateData storage rateData,uint256
amount,address depositor) internal returns (uint256 amountMinted) {
    if (amount < 1) revert InvalidAmount();

    // Transfer asset from caller to the RToken contract
    IERC20(reserve.reserveAssetAddress).safeTransferFrom(
        msg.sender,                    // from
        reserve.reserveRTokenAddress, // to
        amount                         // amount
    );

    // Update reserve interests
```

```
    updateReserveInterests(reserve, rateData);

    // Mint RToken to the depositor (scaling handled inside RToken)
    (bool isFirstMint, uint256 amountScaled, uint256 newTotalSupply, uint256
amountUnderlying) = IRToken(reserve.reserveRTokenAddress).mint(
        address(this),         // caller
        depositor,             // onBehalfOf
        amount,                // amount
        reserve.liquidityIndex // index
    );

    // Update the total liquidity and interest rates
    updateInterestRatesAndLiquidity(reserve, rateData, amount, 0);

    emit Deposit(depositor, amount, amountScaled);

    return amountScaled;
}
```

Severity and Impact Summary

The code could be vulnerable to re-entrancy attacks.

Recommendation

Make sure to follow the Checks-Effects-Interactions pattern.

Governance contract assumptions

Finding ID: FYEO-RAAC-13
Severity: **Informational**
Status: **Remediated**

Description

The Governance contract assumes several things about the token used including the fact it is time locked and has a certain amount of decimals. The token can also be modified while votes are live.

Proof of Issue

File name: contracts/core/governance/Governance.sol
Line number: 163

```
function setVeRAACToken(address _veRAACToken) external onlyOwner {
    veRAACToken = IERC20(_veRAACToken);
```

Severity and Impact Summary

The contract may not run as intended if mistakes are made as they are not prevented in the code.

Recommendation

While these assumptions will likely hold true, it would be better to add additional check.s

Inconsistent use of updateStateBefore modifier

Finding ID: FYEO-RAAC-14
Severity: **Informational**
Status: **Remediated**

Description

The `updateStateBefore` modifier calls `ReserveLibrary.updateReserveState()` and is sometimes used in conjunction with a direct call or not used even though declared.

Proof of Issue

File name: contracts/core/pools/LendingPool/LendingPool.sol
Line number: 292

```
function deposit(uint256 amount) external nonReentrant whenNotPaused
onlyValidAmount(amount) {
    // Update the reserve state before the deposit
    ReserveLibrary.updateReserveState(reserve, rateData);

...

function withdraw(uint256 amount) external nonReentrant whenNotPaused
onlyValidAmount(amount) {
    // Update the reserve state before the withdrawal
    ReserveLibrary.updateReserveState(reserve, rateData);

...

function depositNFT(uint256 tokenId) external nonReentrant whenNotPaused
updateStateBefore {
    // Manually update state
    ReserveLibrary.updateReserveState(reserve, rateData);

...

function withdrawNFT(uint256 tokenId) external nonReentrant whenNotPaused
updateStateBefore {
    ...

    // Update reserve state
    ReserveLibrary.updateReserveState(reserve, rateData);
```

Severity and Impact Summary

Using the `updateStateBefore` modifier and calling `ReserveLibrary.updateReserveState(reserve, rateData)` will waste gas.

Recommendation

Make sure to consistently use one style and to not call the function twice.

Missing zero checks

Finding ID: FYEO-RAAC-15
Severity: **Informational**
Status: **Acknowledged**

Description

There are several missing zero checks throughout the code base.

Proof of Issue

File name: contracts/core/tokens/RAACToken.sol
Line number: 95

```
function setMinter(address _minter) external onlyOwner {
    minter = _minter

...

function burn(uint256 amount) external {
    uint256 taxAmount = amount.percentMul(burnTaxRate);
```

File name: contracts/core/tokens/LPToken.sol
Line number: 22

```
function setMinter(address newMinter) external onlyOwner {
    _setMinter(newMinter);
```

File name: contracts/zeno/Auction.sol
Line number: 57

```
constructor(
    address _zenoAddress,
    address _usdcAddress,
    address _businessAddress,
```

File name: contracts/core/governance/GaugeController.sol
Line number: 34

```
constructor(address _veRAACToken) Ownable(msg.sender) {
    veRAACToken = IERC20(_veRAACToken);
```

File name: contracts/core/governance/Governance.sol
Line number: 50

```
constructor(address _veRAACToken, address _gaugeController, address initialOwner)
Ownable(initialOwner) {
    veRAACToken = IERC20(_veRAACToken);
    gaugeController = GaugeController(_gaugeController);

function setVeRAACToken(address _veRAACToken) external onlyOwner {
    veRAACToken = IERC20(_veRAACToken);
```

```
function setGaugeController(address _gaugeController) external onlyOwner {
    gaugeController = GaugeController(_gaugeController);
```

File name: contracts/zeno/ZENO.sol
Line number: 31

```
constructor(
    address _usdc,
    ...
) Ownable(_initialOwner) ERC20(_name, _symbol) {
    USDC = IERC20(_usdc);
```

File name: contracts/core/pools/StabilityPool/MarketCreator.sol
Line number: 42

```
constructor(address initialOwner, address _raacToken, address _decrvUSDToken)
Ownable(initialOwner) {
    raacToken = IERC20(_raacToken);
    decrvUSDToken = IERC20(_decrvUSDToken);

...

function redeemFromMarket(uint256 marketId) external nonReentrant {
    ...
    uint256 reward = calculateReward(marketId, amount);

    ...
    raacToken.safeTransfer(msg.sender, reward);
```

The reward may be 0.

File name: contracts/core/pools/StabilityPool/NFTLiquidator.sol
Line number: 45

```
constructor(address _crvUSD, address _nftContract, address initialOwner)
Ownable(initialOwner) {
    crvUSD = IERC20(_crvUSD);
    nftContract = IERC721(_nftContract);

function setStabilityPool(address _stabilityPool) external onlyOwner {
    stabilityPool = _stabilityPool;
```

File name: contracts/core/pools/StabilityPool/StabilityPool.sol
Line number: 101

```
function initialize(
    address _rToken,
    address _deToken,
    address _raacToken,
    address _raacMinter,
    address _crvUSDToken,
    address _lendingPool
) public initializer {
    __Ownable_init(_initialOwner);
    __Pausable_init();
    rToken = IRToken(_rToken);
    deToken = IDEToken(_deToken);
    raacToken = IRAACToken(_raacToken);
```

```
    raacMinter = IRAACMinter(_raacMinter);
    crvUSDToken = IERC20(_crvUSDToken);
    lendingPool = ILendingPool(_lendingPool);

...

function setLiquidityPool(address _liquidityPool) external onlyOwner {
    liquidityPool = _liquidityPool;
```

File name: contracts/core/tokens/veRAACToken.sol
Line number: 37

```
constructor(address _raacToken, address initialOwner) ERC20("Vote-escrowed RAAC",
"veRAAC") Ownable(initialOwner) {
    raacToken = IERC20(_raacToken);
```

File name: contracts/core/primitives/RAACHousePrices.sol
Line number: 40

```
function setOracle(address _oracle) external onlyOwner {
    oracle = _oracle;
```

File name: contracts/core/tokens/DEToken.sol
Line number: 79

```
constructor(
    string memory name,
    string memory symbol,
    address initialOwner,
    address _rTokenAddress
) ERC20(name, symbol) ERC20Permit(name) Ownable(initialOwner) {
    rTokenAddress = _rTokenAddress;
```

File name: contracts/core/tokens/RToken.sol
Line number: 132

```
constructor(
    string memory name,
    string memory symbol,
    address initialOwner,
    address assetAddress
) ERC20(name, symbol) ERC20Permit(name) Ownable(initialOwner) {
    _liquidityIndex = WadRayMath.RAY;
    _assetAddress = assetAddress;
```

File name: contracts/core/tokens/RAACNFT.sol
Line number: 26

```
constructor(address _token, address _housePrices, address initialOwner) ERC721("RAAC
NFT", "RAACNFT") Ownable(initialOwner) {
    token = IERC20(_token);
    raac_hp = IRAACHousePrices(_housePrices);
```

Severity and Impact Summary

By mistake 0 values such as the zero address could be passed in which would lead to unusable deployments.

Recommendation

Make sure to consistently check for zero values.

Tokens remain in auction contract if not sold

Finding ID: FYEO-RAAC-16
Severity: **Informational**
Status: **Acknowledged**

Description

There is no clue as to what happens with any potentially remaining tokens after an auction ends.

Proof of Issue

File name: contracts/zeno/Auction.sol
Line number: 15

```
function checkAuctionEnded() external {
    require(block.timestamp >= auctionEndTime, "Auction not ended");
    uint256 price = getPrice();
    emit AuctionEnded(price);
}
```

Severity and Impact Summary

Unclear, as there is no obvious intended functionality.

Recommendation

Make sure this code is implemented according to specifications.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations