Pashov Audit Group

RWf(x) Security Review

November 27th 2025 - December 1st 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of 40+ freelance security researchers, who are well proven in the space - most have earned over $100k in public contest rewards, are multi-time champions or have truly excelled in audits with us. We only work with proven and motivated talent.

With over 300 security audits completed — uncovering and helping patch thousands of vulnerabilities — the group strives to create the absolute very best audit journey possible. While 100% security is never possible to guarantee, we do guarantee you our team's best efforts for your project.

Check out our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users
- **Medium** - leads to a moderate material loss of assets in the protocol or moderately harms a group of users
- **Low** - leads to a minor material loss of assets in the protocol or harms a small group of users

Likelihood

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

4. About RWf(x)

RWf(x) is a protocol that uses RWA-backed tokens like fractionalized gold (fGOLD) as collateral to mint stablecoins (fToken) and leveraged tokens (xToken). It enables splitting yield-bearing assets into a stable, yield-backed coin (goldUSD) and a leveraged asset (xGOLD), balancing stability and exposure to volatility.

5. Executive Summary

A time-boxed security review of the **RegnumAurumAcquisitionCorp/fx-contracts** and **RegnumAurumAcquisitionCorp/fx-contracts** repositories was done by Pashov Audit Group, during which **Tejas Warambhe**, **aslanbek**, **dhank**, **Dimulski**, **t.aksoy** engaged to review **RWf(x)**. A total of **4** issues were uncovered.

Protocol Summary	
Project Name	RWf(x)
Protocol Type	RWA Tokenization
Timeline	November 27th 2025 - December 1st 2025

Review commit hashes:

- 7b77d6982bfb4b70515d8bd245b753589c6bf07d (RegnumAurumAcquisitionCorp/fx-contracts)
- af531c8b0b96627fcfc0b0c2510db63e3a3947be (RegnumAurumAcquisitionCorp/fx-contracts)

Fixes review commit hashes:

- e9bb70de7b0cd456317fe944c3465b64aa0d668d (RegnumAurumAcquisitionCorp/fx-contracts)
- daa893be7fa11067a4517455122d5e993dfde4cc (RegnumAurumAcquisitionCorp/fx-contracts)

Scope

`FxLowVolatilityMath.sol` `FractionalToken.sol` `HarvestableTreasury.sol` `LeveragedToken.sol` `Market.sol` `Treasury.sol` `TokenBlender.sol` `ChainlinkOracleAdapter.sol`

6. Findings

Findings count

Severity	Amount
Low	4
Total findings	**4**

Summary of findings

ID	Title	Severity	Status
[L-01]	Missing deadline check in swap	Low	Acknowledged
[L-02]	Protocol reset upon `totalBaseToken == 0` can lead to cascading issues	Low	**Resolved**
[L-03]	Emergency prices can never reset	Low	Acknowledged
[L-04]	`Treasury.maxMintableXToken()` reverts with div by zero when `xNav` is zero	Low	Acknowledged

Low findings

[L-01] Missing deadline check in swap

TokenBlender#swap lacks the deadline check. Therefore, the swap can be executed when it is not desirable by the caller anymore. Consider adding a deadline check.

[L-02] Protocol reset upon `totalBaseToken == 0` can lead to cascading issues

The `Treasury::totalBaseToken` variable keeps a track of all the base asset tokens in the system. Let's consider a scenario where the protocol is currently functioning as intended: 1. The `totalBaseToken` is currently greater than 0. 2. Due to unforeseen circumstances, the collateral ratio is drops below `1`. 3. During such a situation, the `fTokens` can be redeemed on a pro-rata basis. 4. If all the `fTokens` are redeemed, the `totalBaseToken` will be marked as `0`, but xTokens are still available with the users. 5. The protocol will reset as soon as the `totalBaseToken` hits 0, as the `_loadSwapState()` resets the `xNav` and `fNav`:

```
  function _loadSwapState() internal view returns (FxLowVolatilityMath.SwapState memory _state)
{
    _state.baseSupply = totalBaseToken;
    _state.baseNav = _fetchRWAOraclePrice();

    if (_state.baseSupply == 0) {
      _state.fNav = PRECISION;            <<@
      _state.xNav = PRECISION;           <<@
    } else {
```

1. During this state, only the `Treasury::mint()` with mint option as `Both` can be invoked.
2. However, minting at this point would mean that the collateral ratio is reset, but this will actually subsidise the xToken holders of the past, which is unfair for the new minters.
3. `xToken` holders can simply redeem these `xTokens` on a better rate by just waiting for the protocol to reset.

Hence, it can be observed that the current implementation lacks a way to handle scenarios of complete redemptions.

Recommendations

It is recommended to completely isolate the current system in case of a protocol reset and use a new deployment instead to discourage the use of old `xTokens`.

[L-03] Emergency prices can never reset

The `ChainlinkOracleAdapter` contract's constructor checks for `_emergencyPrice == 0`, which signifies that an emergency price amount must be set:

```
    constructor(address _priceFeed, uint256 _maxStaleTime, uint256 _emergencyPrice, uint256
_priceTolerance, address _initialOwner) Ownable(_initialOwner) {
        if (_priceFeed == address(0)) revert InvalidAddress();
        if (_maxStaleTime == 0 || _priceTolerance == 0 || _emergencyPrice == 0) revert
ZeroValue();      <<@
        priceFeed = AggregatorV3Interface(_priceFeed);
        maxStaleTime = _maxStaleTime;
        emergencyPrice = _emergencyPrice;          <<@
        priceTolerance = _priceTolerance;
    }
```

The code is designed to revert whenever an emergency price is not set:

```
    function getPrice() external view override returns (bool isValid, uint256 price, uint256
minUnsafePrice, uint256 maxUnsafePrice) {
        uint256 safePrice;

        try priceFeed.latestRoundData() returns (uint80, int256 answer, uint256, uint256
updatedAt, uint80) {
            if (answer <= 0) {
                if (emergencyPrice == 0) revert EmergencyPriceNotSet();          <<@
                safePrice = emergencyPrice;
                isValid = false;
            } else {
                uint8 feedDecimals = priceFeed.decimals();
                uint256 chainlinkPrice = _normalizeTo8Decimals(uint256(answer), feedDecimals);

                uint256 age = block.timestamp > updatedAt ? block.timestamp - updatedAt : 0;
                if (age > maxStaleTime) {
                    if (emergencyPrice == 0) revert EmergencyPriceNotSet();         <<@
                    safePrice = emergencyPrice;
                    isValid = false;
                } else {
                    safePrice = chainlinkPrice;
                    isValid = true;
                }
            }
        } catch {
            if (emergencyPrice == 0) revert EmergencyPriceNotSet();         <<@
            safePrice = emergencyPrice;
            isValid = false;
        }

        price = safePrice;

        uint256 priceToleranceValue = safePrice * priceTolerance / BPS_PRECISION;
        minUnsafePrice = safePrice > priceToleranceValue ? safePrice - priceToleranceValue : 0;
        maxUnsafePrice = safePrice + priceToleranceValue;
    }
```

However, the `setEmergencyPrice()` function never allows the owner to reset the emergency price:

```
  function setEmergencyPrice(uint256 _emergencyPrice) external onlyOwner {
      if (_emergencyPrice == 0) revert ZeroValue();        <<@
      emergencyPrice = _emergencyPrice;
      emit EmergencyPriceUpdated(_emergencyPrice);
  }
```

It indicates that the `EmergencyPriceNotSet()` error is an unreachable condition. This is incorrect because emergency prices should be set and observed during a case of emergency; it is practically not possible for the protocol to set a fixed pre-emptive emergency price, and reverting the oracle is a better choice to ensure that the prices are set manually during such situations.

Recommendations

It is recommended to remove the `_emergencyPrice == 0` value altogether:

```
  function setEmergencyPrice(uint256 _emergencyPrice) external onlyOwner {
-        if (_emergencyPrice == 0) revert ZeroValue();
      emergencyPrice = _emergencyPrice;
      emit EmergencyPriceUpdated(_emergencyPrice);
  }
```

[L-04] `Treasury.maxMintableXToken()` reverts with div by zero when `xNav` is zero

https://github.com/RegnumAurumAcquisitionCorp/fx-contracts/blob/2716a80e3ca884738b5998a8613dca3c29789612/contracts/f(x)/math/FxLowVolatilityMath.sol#L110

```
    function maxMintableXToken(SwapState memory state, uint256 _newCollateralRatio)
  internal
  pure
  returns (uint256 _maxBaseIn, uint256 _maxXTokenMintable)
{
  uint256 _baseVal = state.baseNav.mul(state.baseSupply).mul(PRECISION);
  uint256 _fVal = _newCollateralRatio.mul(state.fSupply).mul(state.fNav);


  if (_fVal > _baseVal) {
    uint256 _delta = _fVal - _baseVal;


    _maxBaseIn = _delta.div(state.baseNav.mul(PRECISION));
=@1>    _maxXTokenMintable = _delta.div(state.xNav.mul(PRECISION));
    }
  }
```

- `Market.addBaseToken()` calls `treasury.maxMintableXToken(marketConfig.stabilityRatio)` here to find the `_maxBaseInBeforeSystemStabilityMode` .
- `Treasury._loadSwapState()` sets `_state.xNav = 0` when `_fVal > _baseVal` . Calls _state.maxMintableXToken(_newCollateralRatio). here
- Since `current CR < stabilityRatio` and `_fVal > _baseVal` -> enters `if` block of the above code snippet.
- At this line ,_maxXTokenMintable = _delta.div(0 * PRECISION) -> division by 0 -> **revert**

BaseTokens cannot be added in System Stability Mode or when underCollatarized.

Recommendations

```
    function maxMintableXToken(SwapState memory state, uint256 _newCollateralRatio)
    internal
    pure
    returns (uint256 _maxBaseIn, uint256 _maxXTokenMintable)
  {
    uint256 _baseVal = state.baseNav.mul(state.baseSupply).mul(PRECISION);
    uint256 _fVal = _newCollateralRatio.mul(state.fSupply).mul(state.fNav);


    if (_fVal > _baseVal) {
      uint256 _delta = _fVal - _baseVal;


      _maxBaseIn = _delta.div(state.baseNav.mul(PRECISION));
+       if(state.xNav !=0 )  //else set it 0
       _maxXTokenMintable = _delta.div(state.xNav.mul(PRECISION));
    }
  }
```