Pashov Audit Group

RWf(x) Security Review

November 17th 2025 - November 20th 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** repository was done by Pashov Audit Group, during which **t.aksoy**, **BenRai**, **Nyx**, **jesjupyter** engaged to review **RWf(x)**. A total of **4** issues were uncovered.

Protocol Summary	
Project Name	RWf(x)
Protocol Type	RWA Tokenization
Timeline	November 17th 2025 - November 20th 2025

Review commit hash:

- 7e4b9108261905012ddbaeb3eb4168a51fa33776 (RegnumAurumAcquisitionCorp/fx-contracts)

Fixes review commit hash:

- 7f441e450875bc3fc0de571f4979e81643033040 (RegnumAurumAcquisitionCorp/fx-contracts)

Scope

`FxLowVolatilityMath.sol` `FractionalToken.sol` `HarvestableTreasury.sol` `LeveragedToken.sol` `Market.sol` `Treasury.sol` `IFxFractionalToken.sol` `IFxMarket.sol` `IFxTreasury.sol`

6. Findings

Findings count

Severity	Amount
Low	4
Total findings	**4**

Summary of findings

ID	Title	Severity	Status
[L-01]	`maxMintableXToken()` reverts if system is under-collateralized	Low	Acknowledged
[L-02]	Validation missing for stability ratio vs initial mint ratio	Low	Acknowledged
[L-03]	Inconsistent check for `stabilityMode`	Low	Acknowledged
[L-04]	Over strict check can block minting	Low	**Resolved**

Low findings

[L-01] `maxMintableXToken()` reverts if system is under-collateralized

When the system is under-collateralized, _loadSwapState() sets: `xNav = 0;` But maxMintableXToken() later does: `_maxXTokenMintable = _delta / (xNav * 1e18);` This becomes division by zero, causing the function to revert.

Add a zero check for xNav value and return 0 instead of directly reverting.

[L-02] Validation missing for stability ratio vs initial mint ratio

The `updateMarketConfig` function lacks validation to ensure `stabilityRatio` is less than or equal to `1/initialMintRatio`. When the first mint occurs (when `baseSupply == 0`), the Treasury calculates the collateral ratio as `1/initialMintRatio` based on the initial mint distribution. If `stabilityRatio` is set too high (≥ `1/initialMintRatio`), the system immediately enters stability mode after the first mint, preventing normal operation. This is particularly problematic if `mintBothInSystemStabilityModePaused` is enabled, as it would block all subsequent mints. The function should add a check requiring `_stabilityRatio < PRECISION / treasury.initialMintRatio()` to prevent this configuration issue.

[L-03] Inconsistent check for `stabilityMode`

Through out the code base, the check if the system is in `stabilityMode` is inconsistent. Based on the check in "Market.addBaseToken()" (original forked code), the protocol is in `stabilityMode` when the current `collateralRatio` is below `stabilityRatio`:

```
require(
      _marketConfig.recapRatio <= _collateralRatio && _collateralRatio <
_marketConfig.stabilityRatio,
      "Not system stability mode"
    );
```

Also, all calculations for max tokens that can be minted use `stabilityRatio` as the target ratio to not go into `stabilityMode`:

```
_treasury.maxMintableFToken(marketConfig.stabilityRatio);

_treasury.maxMintableXToken(marketConfig.stabilityRatio);

_treasury.maxRedeemableFToken(_marketConfig.stabilityRatio);

_treasury.maxRedeemableXToken(_marketConfig.stabilityRatio)
```

In other places of the code base, the system is in `stabilityMode` (pauses actions) when `collateralRatio = stabilityRatio`:

Market.mint(): `solidity if (mintBothInSystemStabilityModePaused && _treasury.totalBaseToken() > 0) { uint256 _collateralRatio = _treasury.collateralRatio(); @> require(_collateralRatio > marketConfig.stabilityRatio, "mintBoth paused"); }`

Market.mintFToken():

```
if (fTokenMintInSystemStabilityModePaused) {
      uint256 _collateralRatio = _treasury.collateralRatio();
@>    require(_collateralRatio > marketConfig.stabilityRatio, "fToken mint paused");
```

Market.mintFTokenWithoutBaseToken():

```
if (fTokenMintInSystemStabilityModePaused) {
      uint256 _collateralRatio = _treasury.collateralRatio();
@>    require(_collateralRatio > marketConfig.stabilityRatio, "fToken mint paused");
    }
```

Market.redeem():

```
if (xTokenRedeemInSystemStabilityModePaused) {
        uint256 _collateralRatio = _treasury.collateralRatio();
@>      require(_collateralRatio > _marketConfig.stabilityRatio, "xToken redeem paused");
```

Make sure that the checks for stability mode are consistent.

[L-04] Over strict check can block minting

Market.mint() requires the collateral ratio to be strictly greater than the configured stability ratio:

```
if (mintBothInSystemStabilityModePaused && _treasury.totalBaseToken() > 0) {
      uint256 _collateralRatio = _treasury.collateralRatio();
      require(_collateralRatio > marketConfig.stabilityRatio, "mintBoth paused");

    }
```

Proportional minting keeps the collateral ratio unchanged. Therefore, forbidding mint at exact equality (collateralRatio == stabilityRatio) can unnecessarily revert the mintBoth flow at the threshold.

Recommendations

Consider allowing equality at the threshold, since proportional minting does not decrease the collateral ratio.