0VIX Protocol Audit

May 09, 2022

WATCHPUG

Table of Contents

Summary

This report has been prepared for **0VIX Protocol** smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**0VIX Protocol**
Codebase	https://github.com/0Vix/0vix-protocol
Commit	f6c28b110878e068f8a24d9e122194503ff0b070
Language	Solidity
Platform	Polygon

Audit Summary

Delivery Date	**May 09, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**10**

VX-H1: Wrong values for PriceData when updating prices[oToken]

High

In OvixChainlinkOracleV2.sol#setUnderlyingPrice(), prices[oToken] = PriceData(underlyingPriceMantissa, updatedAt); should be prices[oToken] = PriceData(updatedAt, underlyingPriceMantissa);.

chainlink/OvixChainlinkOracleV2.sol#L96-L114

```
function setUnderlyingPrice(
    address oToken,
    uint underlyingPriceMantissa,
    uint256 updatedAt
) external onlyAdmin {
    require(underlyingPriceMantissa > 0, "bad price");
    if (block.timestamp > updatedAt) {
        // reject stale price
        // validPeriod can be set to 5 mins
        require(block.timestamp - updatedAt < validPeriod, "bad updatedAt");
    } else {
        // reject future timestamp (< 3s is allowed)
        require(updatedAt - block.timestamp < 3, "bad updatedAt");
        updatedAt = block.timestamp;
    }

    emit PricePosted(oToken, prices[oToken].price, underlyingPriceMantissa, updatedAt);
    prices[oToken] = PriceData(underlyingPriceMantissa, updatedAt);
}
```

See the definition of struct PriceData:

```
struct PriceData {
    uint256 updatedAt;
    uint256 price;
}
```

Status

✓ Fixed in commit: 26c0e21ede6b2acffa000616cec151ccccdf93ab.

VX-H2: Actual total boosted rewards can be higher than expected

High

vote-escrow/BoostManager.sol#L205-L223

```
// marketType: 0 = supply, 1 = borrow
// boost basis = totalVeSupply/marketLiquidity
function calcBoostBasis(address market, uint256 marketType)
    internal
    view
    returns (uint256)
{
    require(marketType <= 1, "wrong market type");

    if (marketType == 0) {
        if (IOToken(market).totalSupply() == 0) return 0;
        return ((veOVIX.totalSupply() * MULTIPLIER) /
            IOToken(market).totalSupply());
    } else {
        if (IOToken(market).totalBorrows() == 0) return 0;
        return ((veOVIX.totalSupply() * MULTIPLIER) /
            IOToken(market).totalBorrows());
    }
}
```

vote-escrow/BoostManager.sol#L57-L75

```
/**
    * @notice Updates the boost basis of the user with the latest veBalance
    * @param user Address of the user which booster needs to be updated
    * @return The boolean value indicating whether the user still has the booster greater than 1.0
    */
function updateBoostBasis(address user)
    external
    onlyAuthorized
    returns (bool)
{
    IOToken[] memory markets = comptroller.getAllMarkets();

    veBalances[user] = veOVIX.balanceOf(user);
    for (uint256 i = 0; i < markets.length; i++) {
        _updateBoostBasisPerMarket(address(markets[i]), user);
    }

    return veBalances[user] == 0 ? false : true;
}
```

Comptroller.sol#L1485-L1491

```
function updateAndDistributeSupplierRewardsForToken(
    address oToken,
    address account
) public override {
    updateRewardSupplyIndex(oToken);
    distributeSupplierReward(oToken, account);
}
```

Comptroller.sol#L1571-L1612

```
function distributeSupplierReward(address oToken, address supplier)
    internal
{
    // TODO: Don't distribute supplier Reward if the user is not in the supplier market.
    // This check should be as gas efficient as possible as distributeSupplierReward is called in many places.
    // - We really don't want to call an external contract as that's quite expensive.

    MarketState storage supState = supplyState[oToken];
    uint256 supplyIndex = supState.index;
    uint256 supplierIndex = rewardSupplierIndex[oToken][supplier];

    // Update supplier's index to the current index since we are distributing accrued VIX
    rewardSupplierIndex[oToken][supplier] = supplyIndex;

    if (supplierIndex == 0 && supplyIndex >= 0) {
        supplierIndex = supplyIndex;
    }

    // Calculate change in the cumulative sum of the Reward per oToken accrued
    Double memory deltaIndex = Double({
        mantissa: supplyIndex - supplierIndex
    });

    uint256 supplierTokens = address(boostManager) == address(0)
        ? IOToken(oToken).balanceOf(supplier)
        : boostManager.boostedSupplyBalanceOf(oToken, supplier);

    if (supplyIndex != supplierIndex) {
        // Calculate Reward accrued: oTokenAmount * accruedPerOToken
        uint256 supplierDelta = mul_(supplierTokens, deltaIndex);

        uint256 supplierAccrued = rewardAccrued[supplier] + supplierDelta;
        rewardAccrued[supplier] = supplierAccrued;

        emit DistributedSupplierReward(
            IOToken(oToken),
            supplier,
            supplierDelta,
            supplyIndex
        );
    }
}
```

Given the ever decreasing nature of veToken, the totalSupply, veBalances, and other derived concepts such as `boostManager.boostedTotalSupply` will most certainly be inaccurate at any given time.

A sophisticated attacker or a malicious user can take advantage of this and take a larger portion of the rewards. And the actual emission can be higher than expected.

PoC

Given:

- oMatic's supply RewardSpeeds: 1e18 wei / sec
- account1: mint for 500e8 oMatic with 10e18 matic
- account2: mint for 500e8 oMatic with 10e18 matic
- account3: mint for 500e8 oMatic with 10e18 matic

And:

- account1: lock 10 0VIX, 1yr
- account2: lock 10 0VIX, 1yr

```
// account1 booster 2.5
// account2 booster 2.5
// account3 booster 1
```

- await increaseTime(WEEK);

(state1)

Case A (the normal case):

When:

- (after state1)
- comptroller.claimRewards([account1.address, account2.address, account3.address], [oMatic.address], true, true)

Then:

- account1 received supply reward: 252002458333333333333333
- account2 received supply reward: 252002458333333333333333
- account3 received supply reward: 100800983333333333333333
- 252002458333333333333333 + 252002458333333333333333 + 100800983333333333333333 = 604805899999999999999999 ≈ WEEK * 1e18

Case B (more emission than expected):

When:

- (after state1)
- await comptroller.updateAndDistributeSupplierRewardsForToken(oMatic.address, owner.address); // update market RewardSupplyIndex
- account3: lock 100 0VIX, 1yr
- await voteController.connect(account3).voteForMarketWeights(oMatic.address, 10000); // account3 boostManager.updateBoostBasis(), account3 booster 1 -> 2.5
- comptroller.claimRewards([account1.address, account2.address, account3.address], [oMatic.address], true, true)

Then:

- account1 received supply reward: 252003791666666666666666
- account2 received supply reward: 252003791666666666666666
- account3 received supply reward: 252003791666666666666666
- 252003791666666666666666 + 252003791666666666666666 + 252003791666666666666666 = 756011374999999999999998 ≈ WEEK * 1e18 + 150000 * 1e18

In this case, the total emission exceeded the expected total emission by 150000 * 1e18.

Status

✓ Fixed in commit: a967163ce198b9db5ff3a76d77c27494e3e8c358.

VX-H3: OvixChainlinkOracle network congestion may disrupt price feeds

High

chainlink/OvixChainlinkOracle.sol#L64-L73

```
function setUnderlyingPrice(IOToken oToken, uint underlyingPriceMantissa) external onlyAdmin {
    address asset = address(OErc20(address(oToken)).underlying());
    emit PricePosted(asset, prices[asset], underlyingPriceMantissa, underlyingPriceMantissa);
    prices[asset] = underlyingPriceMantissa;
}

function setDirectPrice(address asset, uint price) external onlyAdmin {
    emit PricePosted(asset, prices[asset], price, price);
    prices[asset] = price;
}
```

The implementation only takes two parameters: the token and the price. The time of the price is not included.

This makes it possible for the price feeds to be disrupted when the network is congested and transactions with stale prices get accepted as fresh prices.

Since the price feeds are essential to the protocol, that can result in users' positions being liquidated wrongfully and case fund loss to users.

PoC

Given:

- admin of OvixChainlinkOracle.sol is connected to an RPC endpoint currently experiencing degraded performance;
- Bitcoin price is $100,000;
- The collateralFactor of Bitcoin is 60%.

1. Alice borrowed 50,000 USDC with 1 BTC as collateral;
2. Bitcoin price dropped to $90,000, to avoid liquidation, Alice repaid 10,000 USD;
3. The price of Bitcoin dropped to $80,000; admin of OvixChainlinkOracle.sol tries to setDirectPrice() with the latest price: $80,000, however, since the network is congested, the transaction was not get mined timely;
4. Bitcoin price rebound to $100,000; Alice borrowed another 10,000 USDC;
5. The tx send by admin at step 3 finally got mined, the protocol now believes the price of Bitcoin has suddenly dropped to $80,000, as a result, Alice's position got liquidated.

Recommendation

Change to:

```
function setDirectPrice(address _asset, uint _price, uint _updatedAt) external onlyAdmin {
  require(_price > 0, "bad price");
  if (block.timestamp > _updatedAt) {
      // reject stale price
      // validPeriod can be set to 5 mins
      require(block.timestamp - _updatedAt < validPeriod, "bad updatedAt");
  } else {
      // reject future timestamp (< 3s is allowed)
      require(_updatedAt - block.timestamp < 3, "bad updatedAt");
      _updatedAt = block.timestamp;
  }

  emit PricePosted(_asset, prices[_asset].prce, _price, _price);

  prices[_asset] = PriceData({
    price: Math.safe216(_price),
    updatedAt: Math.safe32(_updatedAt)
  });
}
```

Status

✓ **Fixed** in commit: a1f9579938cb211e75dd5727054d7c5782396d6a.

VX-M4: Improper handling of stale self-updated price

Medium

chainlink/OvixChainlinkOracleV2.sol#L55-L71

```
function getPrice(IOToken oToken) internal view returns (uint price) {
    IEIP20 token = IEIP20(OErc20(address(oToken)).underlying());

    if (prices[address(oToken)].price != 0) {
        price = prices[address(oToken)].price;
    } else {
        price = getChainlinkPrice(getFeed(address(oToken)));
    }

    uint decimalDelta = uint(18).sub(uint(token.decimals()));
    // Ensure that we don't multiply the result by 0
    if (decimalDelta > 0) {
        return price.mul(10**decimalDelta);
    } else {
        return price;
    }
}
```

In `getPrice()`, when a self-updated price exists, it should check the updated time of the price and see if it's stale before using it.

Also, consider using Chainlink's feed first if possible.

Recommendation

Change to:

```
function getPrice(IOToken oToken) internal view returns (uint price) {
    IEIP20 token = IEIP20(OErc20(address(oToken)).underlying());

    IAggregatorV2V3 feed = getFeed(address(oToken));
    if (feed) {
        price = getChainlinkPrice(feed);
    } else if (prices[address(oToken)].updatedAt >= block.timestamp - validPeriod) {
        price = prices[address(oToken)].price;
    }

    require(price > 0, "bad price");

    uint decimalDelta = uint(18).sub(uint(token.decimals()));
    // Ensure that we don't multiply the result by 0
    if (decimalDelta > 0) {
        return price.mul(10**decimalDelta);
    } else {
        return price;
    }
}
```

Status

✓ **Fixed** in commit: 5b5b3c69ce97b01e8e8048e609a217a51a2d4903.

VX-M5: OvixChainlinkOracle.sol#getPrice() Incompatible with some tokens

Medium

Some ERC20 tokens, for example MakerDAO's MKR token, is not using string for token name and symbol.

As a result, IEIP20().symbol() will revert.

Furthurmore, symbol() is not reliable to be the identity of the token.

chainlink/OvixChainlinkOracle.sol#L35-L51

```
function getPrice(IOToken oToken) internal view returns (uint price) {
        IEIP20 token = IEIP20(OErc20(address(oToken)).underlying());

        if (prices[address(token)] != 0) {
            price = prices[address(token)];
        } else {
            price = getChainlinkPrice(getFeed(token.symbol()));
        }

        uint decimalDelta = uint(18).sub(uint(token.decimals()));
        // Ensure that we don't multiply the result by 0
        if (decimalDelta > 0) {
            return price.mul(10**decimalDelta);
        } else {
            return price;
        }
    }
```

Status

✓ Fixed in commit: a1f9579938cb211e75dd5727054d7c5782396d6a.

VX-M6: Use of deprecated Chainlink function latestAnswer

Medium

chainlink/OvixChainlinkOracle.sol#L53-L62

```
function getChainlinkPrice(IAggregatorV2V3 feed) internal view returns (uint) {
    // Chainlink USD-denominated feeds store answers at 8 decimals
    uint decimalDelta = uint(18).sub(feed.decimals());
    // Ensure that we don't multiply the result by 0
    if (decimalDelta > 0) {
        return uint(feed.latestAnswer()).mul(10**decimalDelta);
    } else {
        return uint(feed.latestAnswer());
    }
}
```

According to Chainlink's documentation, the `latestAnswer` function is deprecated. This function does not error if no answer has been reached but returns 0, causing an incorrect price fed to `OvixChainlinkOracle`.

Recommendation

Use the `latestRoundData` method instead.

See: https://docs.chain.link/docs/historical-price-data/#solidity

Status

✓ Fixed in commit: a1f9579938cb211e75dd5727054d7c5782396d6a.

VX-M7: Wrong implementation makes the emission rate 2x than expected

Medium

The totalEmissions usually represent the total emission rate, and it will be distributed among the markets according to a certain set of rules.

However, in the current implementation, after the reward for a specific market is calculated, both the supply and borrow side will get the 100% of the reward, making each market get 2x the reward they are expected to get.

As a result, the actual total emission rate is 2x the totalEmissions.

vote-escrow/VoteController.sol#L664-L699

```
function updateRewards() public {
    require(
        block.timestamp >= nextTimeRewardsUpdated,
        "rewards already updated"
    );
    checkpointAll();
    nextTimeRewardsUpdated = ((block.timestamp + PERIOD) / PERIOD) * PERIOD;
    uint256 votableAmount = (totalEmissions * votablePercentage) /
        HUNDRED_PERCENT;
    uint256 fixedAmount = totalEmissions - votableAmount;

    for (uint256 i = 0; i < markets.length(); i++) {
        // todo: check if all markets have (fixed-)weights
        address addr = markets.at(i);
        uint256 relWeight = _marketRelativeWeight(addr, block.timestamp);
        uint256 reward = ((fixedAmount *
            fixedRewardWeights[markets.at(i)]) / HUNDRED_PERCENT) +
            ((votableAmount * relWeight) / 1e18);

        address[] memory addrs = new address[](1);
        addrs[0] = addr;

        uint256[] memory rewards = new uint256[](1);
        rewards[0] = reward;

        // current implementation doesn't differentiate supply and borrow reward speeds
        comp._setRewardSpeeds(addrs, rewards, rewards);
        updates.push(Updated(addr, reward, block.timestamp));
        emit RewardsUpdated(addr, reward, reward, fixedRewardWeights[markets.at(i)], relWeight);
    }

    // shift the epoch so the booster of the needed users can be decreased
    shiftingEpoch = shiftingEpoch == Epoch.THIRD
        ? Epoch.FIRST
        : Epoch(uint256(shiftingEpoch) + 1);
}
```

Recommendation

The simpliest fix is change to:

```
comp._setRewardSpeeds(addrs, rewards / 2, rewards / 2);
```

A more complete and flexible resolution is, to have another storage mapping called `marketSupplySideBPS`:

```
// marketAddr -> supplySideBPS
mapping(address => uint256) public marketSupplySideBPS;
```

```
uint256 reward = ((fixedAmount *
    fixedRewardWeights[markets.at(i)]) / HUNDRED_PERCENT) +
    ((votableAmount * relWeight) / 1e18);

uint supplySideRewards = reward * marketSupplySideBPS[addr] / MAX_BPS;
comp._setRewardSpeeds(addrs, [supplySideRewards], [reward - supplySideRewards]);
```

Status

✓ **Fixed** in commit: 25c95c3704b2a9043768873d6a1cb7e4f751d048.

VX-L8: symbol() should not be used as the identity of the token

Low

ERC20's symbol() is a optional method, and there are plenty of tokens using the same symbol.

Using the symbol() as the identity can potentially disrupt the price feed in the case that we added a second token with the same symbol into the system.

chainlink/OvixChainlinkOracle.sol#L75-L83

```
function setFeed(string calldata symbol, address feed) external onlyAdmin {
    require(feed != address(0) && feed != address(this), "invalid feed address");
    emit FeedSet(feed, symbol);
    feeds[keccak256(abi.encodePacked(symbol))] = IAggregatorV2V3(feed);
}

function getFeed(string memory symbol) public view returns (IAggregatorV2V3) {
    return feeds[keccak256(abi.encodePacked(symbol))];
}
```

chainlink/OvixChainlinkOracle.sol#L35-L51

```
function getPrice(IOToken oToken) internal view returns (uint price) {
        IEIP20 token = IEIP20(OErc20(address(oToken)).underlying());

        if (prices[address(token)] != 0) {
            price = prices[address(token)];
        } else {
            price = getChainlinkPrice(getFeed(token.symbol()));
        }

        uint decimalDelta = uint(18).sub(uint(token.decimals()));
        // Ensure that we don't multiply the result by 0
        if (decimalDelta > 0) {
            return price.mul(10**decimalDelta);
        } else {
            return price;
        }
    }
```

Status

✓ Fixed in commit: 5b5b3c69ce97b01e8e8048e609a217a51a2d4903.

VX-L9: Shadowing variables

Low

Comptroller.sol#L1273-L1296

```
function _initializeMarket(address oToken) internal {
    uint32 timestamp = safe32(getTimestamp());

    MarketState storage supplyState = supplyState[oToken];
    MarketState storage borrowState = borrowState[oToken];

    /*
        * Update market state indices
        */
    if (supplyState.index == 0) {
        // Initialize supply state index with default value
        supplyState.index = marketInitialIndex;
    }

    if (borrowState.index == 0) {
        // Initialize borrow state index with default value
        borrowState.index = marketInitialIndex;
    }

    /*
        * Update market state timestamps
        */
    supplyState.timestamp = borrowState.timestamp = timestamp;
}
```

`supplyState` and `borrowState` are shadowing local variables.

Comptroller.sol#L1535-L1560

```
function updateRewardBorrowIndex(
    address oToken,
    Exp memory marketBorrowIndex
) internal {
    MarketState storage borrowState = borrowState[oToken];
    uint256 borrowSpeed = rewardBorrowSpeeds[oToken];
    uint32 timestamp = safe32(getTimestamp());
    uint256 deltaBlocks = uint256(timestamp) -
        uint256(borrowState.timestamp);
    if (deltaBlocks > 0) {
        if (borrowSpeed > 0) {
            uint256 borrowAmount = div_(
                    address(boostManager) == address(0) ? IOToken(oToken).totalBorrows() :
boostManager.boostedTotalBorrows(oToken),
                marketBorrowIndex
            );
            uint256 rewardAccrued = deltaBlocks * borrowSpeed;
            Double memory ratio = borrowAmount > 0
                ? fraction(rewardAccrued, borrowAmount)
                : Double({mantissa: 0});
            borrowState.index = safe224(
                add_(Double({mantissa: borrowState.index}), ratio).mantissa
            );
        }
        borrowState.timestamp = timestamp;
    }
}
```

`borrowState` is shadowing local variables.

Comptroller.sol#L1567-L1580

```
function distributeSupplierReward(address oToken, address supplier)
    internal
{
    // TODO: Don't distribute supplier Reward if the user is not in the supplier market.
    // This check should be as gas efficient as possible as distributeSupplierReward is called in many places.
    // - We really don't want to call an external contract as that's quite expensive.

    MarketState storage supplyState = supplyState[oToken];
    uint256 supplyIndex = supplyState.index;
    uint256 supplierIndex = rewardSupplierIndex[oToken][supplier];

    // Update supplier's index to the current index since we are distributing accrued VIX
    rewardSupplierIndex[oToken][supplier] = supplyIndex;
    ...
```

`supplyState` is shadowing local variables.

Comptroller.sol#L1618-L1632

```
function distributeBorrowerReward(
    address oToken,
    address borrower,
    Exp memory marketBorrowIndex
) internal {
    // TODO: Don't distribute supplier Reward if the user is not in the borrower market.
    // This check should be as gas efficient as possible as distributeBorrowerReward is called in many places.
    // - We really don't want to call an external contract as that's quite expensive.

    MarketState storage borrowState = borrowState[oToken];
    uint256 borrowIndex = borrowState.index;
    uint256 borrowerIndex = rewardBorrowerIndex[oToken][borrower];

    // Update borrowers's index to the current index since we are distributing accrued VIX
    rewardBorrowerIndex[oToken][borrower] = borrowIndex;
    ...
```

`borrowState` is shadowing local variables.

Shadowing local variables is naming conventions found in two or more variables that are similar. Although they do not pose any immediate risk to the contract, incorrect usage of the variables is possible and can cause serious issues if the developer does not pay close attention.

Status

✓ **Fixed** in commit: df0d1e5a05d31d7450bb231dfd4813abe484483e.

VX-L10: distributeSupplierReward() can be disrupted by marketInitialIndex update

Low

Comptroller.sol#L1566-L1586

```
function distributeSupplierReward(address oToken, address supplier)
    internal
{
    // TODO: Don't distribute supplier Reward if the user is not in the supplier market.
    // This check should be as gas efficient as possible as distributeSupplierReward is called in many places.
    // - We really don't want to call an external contract as that's quite expensive.

    MarketState storage supplyState = supplyState[oToken];
    uint256 supplyIndex = supplyState.index;
    uint256 supplierIndex = rewardSupplierIndex[oToken][supplier];

    // Update supplier's index to the current index since we are distributing accrued VIX
    rewardSupplierIndex[oToken][supplier] = supplyIndex;

    if (supplierIndex == 0 && supplyIndex >= marketInitialIndex) {
        // Covers the case where users supplied tokens before the market's supply state index was set.
        // Rewards the user with Reward accrued from the start of when supplier rewards were first
        // set for the market.
        supplierIndex = marketInitialIndex;
    }

    // Calculate change in the cumulative sum of the Reward per oToken accrued
    Double memory deltaIndex = Double({
        mantissa: supplyIndex - supplierIndex
    });
```

`if (supplierIndex == 0 && supplyIndex >= marketInitialIndex) {` was `if (supplierIndex == 0 && supplyIndex >= 0) {` in the original Compound code.

We are not sure why the code is changed, but for the current implementation, once `marketInitialIndex` is updated to a higher value in a future version, `distributeSupplierReward()` can be disrupted:

In that case, `supplierIndex` won't be set to `marketInitialIndex` at L1585. As a result, `supplyIndex - supplierIndex` == `supplyIndex - 0` == `supplyIndex`.

The `supplier` will receive a much larger amount of rewards as expected.

The same issue also exists on `distributeBorrowerReward()`.

Recommendation

Change to:

```
if (supplierIndex == 0 || supplierIndex >= supplyIndex) return;
```

Status

✓ **Fixed** in commit: 042ab4658c8b07f1b19881502a62efaf300be131.

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer