Vyper Security Audit

Report Version 0.1

28.10.2024

Conducted by :

Pyro, Security Researcher

Table of Contents

1 About the auditor

Pyro is distinguished independent smart contract security researcher with robust track record. Over the past year, he has improved the security of many protocols, working both alone and with others. Previously with Guardian, Pyro has audited high-profile clients like Synthetix and GMX, earning him a reputation as a trusted blockchain security researcher. Learn more at https://github.com/0x3b33.

2 Disclaimer

Audits are a time, resource, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

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

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

From October 22-27, 2024, Pyro conducted a security review of Vyper.

Overview

Project Name	Vyper
Repository	private
Audit hash	22b9...7289
Remediation	1bee...cb48
Methods	Manual review

Timeline

v0.1	22.10.2024	Audit kick-off
v0.1	27.10.2024	Preliminary report
v0.1	28.10.2024	Mitigation review

Scope

src/Vyper.sol
src/Auction.sol
src/VyperTreasury.sol
src/nexus/DragonXVoltInput.sol
src/nexus/VyperDragonXNexus.sol
src/nexus/VoltVyperNexus.sol
src/nexus/DragonXVoltNexus.sol

Issues Found

Critial risk	2
High risk	1
Medium risk	2
Low risk	4
Informational	4

5 System overview

Vyper is part of the TitanX ecosystem, with its main goal to close the loop between DragonX and Volt. It implements numerous ways to distribute and reward its users while simultaneously burning portions of its token supply.

The focus of the security review was on the following:

1. Nexus module and its enclosed loop
2. Auction and vyper token
3. Fees, token distributions, and burns
4. Epochs and their smaller intervals

6 Findings

6.1 Crit

6.1.1 collectFees transfers vyper instead of volt tokens

Severity: *Crit*

Context: `Auction.sol#L222-L224`

Description:

`collectFees`, used to collect fees from the LP pool, accidentally transfers `vyper` tokens when it should transfer `volt` instead.

This error will prevent any `volt` tokens earned from LP staking from functioning properly. Additionally, it has major effects on the system, as `vyper` is the reward token and is precisely distributed. This issue takes a portion of the rewards and prevents the last users from claiming, as the rewards won't be sufficient.

When the contracts have few rewards left, it would also cause the function to revert, preventing the admin from claiming the generated fees (including `dragonX` and `volt` tokens).

Recommendation:

Change the transferred token.

```
    if (_voltAmount > 0) {
-       vyper.safeTransfer(LIQUIDITY_BONDING_ADDR, _voltAmount);
+       volt.safeTransfer(LIQUIDITY_BONDING_ADDR, _voltAmount);
    }
```

Resolution: Fixed in e83d9ae

6.1.2 Transferring the tokens will be insufficient

Severity: *Crit*

Context: `VoltVyperNexus.sol#L147`

Description:

The functions `swapVoltToVyperAndDistribute` and `swapVyperToDragonXAndDistribute` allocate a specific percentage of each newly swapped token to designated locations. For instance, in the example below, `swapVoltToVyperAndDistribute` allocates 60% to `VyperDragonXNexus`:

```
vyper.transfer(GENESIS_WALLET, wmul(vyperAmount, uint256(0.03e18))); // 3%
vyper.transfer(LIQUIDITY_BONDING_ADDR, wmul(vyperAmount, uint256(0.07e18))); // 7%
vyper.transfer(address(vyper.treasury()), wmul(vyperAmount, uint256(0.2e18))); //
    20%
vyper.transfer(address(vyper.vyperDragonXNexus()), wmul(vyperAmount, uint256(0.6e18)
    )); // 60%
```

However, if `distributeVyperForBurning` is not called, these funds will not be allocated within `toDistribute`, effectively locking them within the contract.

```
function _updateSnapshot() internal {
    if (Time.blockTs() < startTimeStamp || lastSnapshot + 24 hours > Time.blockTs())
         return;

    if (lastSnapshot != 0 && lastSnapshot + 48 hours < Time.blockTs()) {
        // If we have missed an entire snapshot of interactions with the contract
        toDistribute = 0;
    }
    totalVyperDistributed = toDistribute;
    toDistribute = 0;
}
```

Recommendation: Approve the tokens and call the appropriate `distribute` functions to ensure the funds are correctly allocated.

Resolution: Fixed in 4a60961

6.2 High

6.2.1 Not enough dragonX is transferred to dragonXVoltNexus

Severity: *High*

Context: `VyperDragonXNexus.sol#L142`

Description:

In the `swapVyperToDragonXAndDistribute` function, `dragonX` tokens are distributed to `LIQUIDITY_BONDING_ADDR` and `dragonXVoltNexus`. While the split should account for 100% of the amount, both contracts currently receive only 8%, leaving most of the `dragonX` tokens stuck in the contract.

```
dragonX.transfer(LIQUIDITY_BONDING_ADDR, wmul(dragonXAmount, uint256(0.08e18)));
dragonX.transfer(address(vyper.dragonXVoltNexus()), wmul(dragonXAmount, uint256(0.08
    e18)));
```

Recommendation:

Adjust the transfer amounts to properly allocate the remaining percentage:

```
dragonX.transfer(LIQUIDITY_BONDING_ADDR, wmul(dragonXAmount, uint256(0.08e18)));
-   dragonX.transfer(address(vyper.dragonXVoltNexus()), wmul(dragonXAmount, uint256
    (0.08e18)));
+   dragonX.transfer(address(vyper.dragonXVoltNexus()), wmul(dragonXAmount, uint256
    (0.92e18)));
```

Resolution: Fixed in 20c10e4

6.3 Medium

6.3.1 _calculateIntervals will always distribute for 1 less cycle

Severity: *Medium*

Context: `VoltVyperNexus.sol#L279-L283`

Description:

The function `_calculateMissedIntervals` is intended to determine the number of intervals that have passed since the last user interaction. However, for any interval after the first one, it consistently returns `missed intervals - 1`.

```
function _calculateMissedIntervals(uint256 timeElapsedSince) internal view returns (
    uint16 _missedIntervals) {
    _missedIntervals = uint16(timeElapsedSince / INTERVAL_TIME);
    if (lastBurnedIntervalStartTimestamp != 0) _missedIntervals--;
}
```

Within `_calculateIntervals`, this function calculates rewards for all missed intervals. An issue arises because the function only accounts for one fewer interval, as shown below:

Example: - The last call occurred at 13:55; a new call happens at 14:07 (2 intervals have passed).

1. `_calculateMissedIntervals` will return 1 since `(12min / 5min)- 1 = 1`.
2. Upon entering the **`else`** block, `theEndOfTheDay` will be set to 14:00, making `accumulatedIntervalsForTheDay == 1`.
3. The code compares `missedIntervals` and `accumulatedIntervalsForTheDay` and calculates their difference. Since both are 1, the difference is 0.
4. Consequently, no value is added to `_totalAmountForInterval` as `_intervalsForNewDay` is 0.

This results in rewards being calculated for just one cycle. When returning to `getCurrentInterval`, an additional interval is added to `_missedIntervals`, which `_intervalUpdate` later uses to set `lastBurnedIntervalStartTimestamp` to 14:05 (`last + 2 * 5min`), causing `_intervalUpdate` to consistently calculate fewer rewards for burning or distribution.

POC - https://gist.github.com/0x3b33/f335241d5dc7b04c3ceb1d109c209407

Recommendation:

Apply the following adjustment:

```
uint128 _intervalsForNewDay = missedIntervals >= accumulatedIntervalsForTheDay
    ? (missedIntervals - accumulatedIntervalsForTheDay) + 1
    : 0;
```

Resolution: Fixed in 2de7333

6.3.2 Breaching the swapCap will brick some funds inside the contracts

Severity: *Medium*

Context: `VoltVyperNexus.sol#L122-L129`

Description:

All contracts in the nexus module implement a `swapCap`, which limits the amount of tokens allocated for the current snapshot. If `amountAllocated` exceeds `swapCap`, the difference (`amountAllocated - swapCap`) is added to `toDistribute` for the next snapshot.

```
if (currInterval.amountAllocated > swapCap) {
    uint256 difference = currInterval.amountAllocated - swapCap;
    toDistribute += difference;
    currInterval.amountAllocated = swapCap;
}
```

Currently, an issue arises if `swapCap` is breached when moving into the next snapshot. A portion of `difference` becomes locked, with this percentage based on the intervals that have passed for the new day (e.g., passing 10% of intervals for the new day locks 10% of `difference`).

This problem occurs because `intervalUpdate` is triggered before updating `toDistribute`. If moving to the next interval, `_intervalsForNewDay` is calculated, and `toDistribute` is proportionally distributed across intervals, as shown:

```
uint128 _intervalsForNewDay = missedIntervals > accumulatedIntervalsForTheDay
    ? missedIntervals - accumulatedIntervalsForTheDay
    : 0;

_totalAmountForInterval += (_intervalsForNewDay > INTERVALS_PER_DAY)
    ? uint128(toDistribute)
    : uint128(toDistribute / INTERVALS_PER_DAY) * _intervalsForNewDay;
```

Thus, the entirety of `toDistribute` is distributed in intervals, with a set percentage taken each interval. When `toDistribute` increases later, its proportionally locked portion also grows.

Example:

1. `totalVoltDistributed` has accumulated 10k `volt`, and `toDistribute` holds 2k.
2. The admin lowers the `swapCap` to 5k to slow emissions.
3. A user calls `swapVoltToVyperAndDistribute` at 17:00 UTC, 3 hours into the new day.

Upon updating the interval, `amountAllocated` becomes 10k + 2k * 36 / 288 = 10,250. The function then checks `swapCap` and adds 5,250 to `toDistribute`. However, with 12.5% of `toDistribute` already utilized, 656.2 `volt` becomes locked. This affects all nexus modules except `DragonXVoltInput`.

Recommendation:

Call `_updateSnapshot` before increasing `toDistribute` to ensure the increase occurs in the next snapshot:

```
+   _updateSnapshot();
    if (currInterval.amountAllocated > swapCap) {
        uint256 difference = currInterval.amountAllocated - swapCap;
```

```
        toDistribute += difference;
        currInterval.amountAllocated = swapCap;
    }
-   _updateSnapshot();
```

Resolution: Fixed in 5b14ced

6.4 Low

6.4.1 Use safeTransfer

Severity: *Low*

Context: `Auction.sol#L155`

Description:

Description:

Although the tokens follow the EIP-20 standard, it is considered best practice to use `safeTransfer` instead of `transfer` to ensure reliable token transfers.

Recommendation:

Replace all instances of `transfer` with `safeTransfer`:

```
- vyper.transfer(msg.sender, toClaim);
+ vyper.safeTransfer(msg.sender, toClaim);
```

Resolution: Acknowledged

6.4.2 Intervals should be 5 min

Severity: *Low*

Context: `Constants.sol#L26`

Description:

There is a minor inconsistency between the documentation and the code regarding the time interval between buy and burn operations. The documentation specifies a 5-minute interval:

> "5 min intervals that happen at the same time"

However, in the code, the interval is set to 8 minutes:

```
// INTERVALS
uint16 constant INTERVAL_TIME = 8 minutes;
```

Recommendation:

Update either the documentation or the code to ensure consistency. Note that if changing `INTERVAL_TIME` to 5 minutes, `INTERVALS_PER_DAY` would need to be adjusted to prevent overflow, as `24h / 5min = 288` exceeds the capacity of the current `uint8`.

```
-   uint16 constant INTERVAL_TIME = 8 minutes;
-   uint8 constant INTERVALS_PER_DAY = uint8(24 hours / INTERVAL_TIME);

+   uint16 constant INTERVAL_TIME = 5 minutes;
+   uint16 constant INTERVALS_PER_DAY = uint16(24 hours / INTERVAL_TIME);
```

Resolution: Fixed in e83d9ae

6.4.3 LIQUIDITY_BONDING_ADDR is same as GENESIS_WALLET

Severity: *Low*

Context: `Constants.sol#L9`

Description:

The `Constants` file includes standard values, such as percentage distributions and addresses. However, two of the addresses are identical, which could lead to future complications since funds may not be allocated to the expected location.

```
// Distribution addresses
address constant GENESIS_WALLET = 0xfF5758cb7B0F57f332F956A1177a0A0Ed7785eD9;
address constant LIQUIDITY_BONDING_ADDR = 0xfF5758cb7B0F57f332F956A1177a0A0Ed7785eD9
    ; // @todo - TBD //@finding LOW same as GENESIS_WALLET
```

Recommendation:

Consider updating `LIQUIDITY_BONDING_ADDR` to point to the actual liquidity bonding curve address to avoid any potential issues.

Resolution: Fixed in 1bee06d

6.4.4 Protocol has his epoch intervals startTimestamp , which can be different from 2pm UTC

Severity: *Low*

Context: `DeployVyper.s.sol#L34`

Description:

The `Auction` and `DragonXVoltInput` contracts use `startTimestamp` (which can be set to any time) to determine when epochs change. This approach can lead to conflicts with the nexus modules, which utilize 2 PM UTC as a cutoff point and operate on 5-minute intervals.

Recommendation:

Consider changing the start time to align with the nexus modules. This adjustment can be made easily in the deployment scripts:

```
+   import {Time} from "@utils/Time.sol";

        // code...
-   uint32 auctionStartTimestamp = uint32(block.timestamp) + 1 days;
+   uint32 auctionStartTimestamp = getDayEnd(uint32(block.timestamp) + 1 days);
```

Resolution: Acknowledged. We have it in another scripts file that is not present here.

6.5 Informational

6.5.1 Pack storage better

Severity: *Informational*

Context: DragonXVoltInput.sol#L43-L66

Description:

The current storage layout is inefficient, resulting in excessive gas costs during contract deployment. To optimize gas usage, the storage variables should be rearranged to minimize wasted space.

Recommendation:

Adjust the storage layout as shown below to reduce gas costs:

```
    //===========STATE==========//

    /// @notice Timestamp of the last burn call
    uint32 public lastBurnedIntervalStartTimestamp;

+   /// @notice The last burned interval
+   uint256 public lastBurnedInterval;

    /// @notice Maximum amount of Dragon X to be swapped and then burned
    uint128 swapCap;

+   /// @notice Mapping from interval number to Interval struct
+   mapping(uint32 interval => Interval) public intervals;

    /// @notice Last interval number
    uint32 public lastIntervalNumber;

    /// @notice Total DragonX tokens distributed
    uint256 public totalDragonXDistributed;

    /// @notice That last snapshot timestamp
    uint32 lastSnapshot;

-   /// @notice The last burned interval
-   uint256 public lastBurnedInterval;

-   /// @notice Mapping from interval number to Interval struct
-   mapping(uint32 interval => Interval) public intervals;

-   /// @notice Total DragonX tokens distributed
-   uint256 public totalDragonXDistributed;
```

Resolution: Fixed in 854f57c

6.5.2 Remove all unused variables

Severity: *Informational*

Context: `DragonXVoltInput.sol#L52`

Description:

The system includes several values used solely for tracking statistics, primarily for front-end purposes. The `swapCap` variable is one of these, but its state is currently private, making it difficult for the front-end to access its value.

```
    uint128 swapCap;
```

Recommendation:

Consider changing the visibility of `swapCap` to public to facilitate easier access for the front-end:

```
-   uint128 swapCap;
+   uint128 public swapCap;
```

Resolution: Fixed in 5be7d0c

6.5.3 totalVyperBurnt is not used anywhere

Severity: *Informational*

Context: `DragonXVoltNexus.sol#L45`

Description:

In the `DragonXVoltNexus` contract, there is a variable named `totalVyperBurnt` that is initialized but never updated. Similarly, the `FEES_WALLET` variable is declared but never used.

Recommendation:

Consider removing all unused variables to streamline the contract and improve readability.

Resolution: Fixed in a01dbdc

6.5.4 Consider emitting events on important changes

Severity: *Informational*

Context: `Auction.sol#L317-L327`

Description:

The system currently uses very few events, and they are often placed in less significant areas. It is best practice to emit events for important changes, as this can be quite beneficial for the front-end. For example, setting emitted rewards can be significant, particularly since the values will be fixed for the first seven days but unique for each epoch thereafter.

```
    function _updateAuction() internal {
        uint32 daySinceStart = Time.dayGap(startTimestamp, Time.blockTs()) + 1;

        if (dailyStats[daySinceStart].vyperEmitted != 0) return;
        if (daySinceStart > 8 && volt.balanceOf(address(treasury)) == 0) revert
            TreasuryVoltIsEmpty();

        uint256 emitted = daySinceStart <= 8
            ? vyper.mint(address(this), AUCTION_EMIT) // 100e24
            : treasury.emitForAuction();

        dailyStats[daySinceStart].vyperEmitted = uint128(emitted);
    }
```

Recommendation:

Consider emitting events when important variables are set to improve tracking and transparency within the system.

Resolution: Acknowledged