Vyper

Security review

Version 1.0

Reviewed by

deth

nmirchev8

Silvermist

Table of Contents

1 About Egis Security

Egis Security is a team of experienced smart contract researchers, who strive to provide the best smart contract security services possible to DeFi protocols.

The team has a proven track record on public auditing platforms like Code4rena, Sherlock, and Cantina, earning top placements and rewards exceeding $170,000. They have identified over 150 high and medium-severity vulnerabilities in both public contests and private audits.

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 show the presence of vulnerabilities **but not their absence**.

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

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

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

4 Executive summary

Overview

Project Name	Vyper
Repository	Private
Commit hash	1bee06d7dbd163176c24f5feae55a28adb81cb48
Resolution	b7dc32f4e8020e121c24945f9e9a09882edf83bf
Documentation	-
Methods	Manual review

Scope

src/Auction.sol
src/Vyper.sol
src/VyperTreasury.sol
actions/*
const/*
libs/*
nexus/*
utils/*

Issues Found

Critical risk	0
High risk	0
Medium risk	2
Low risk	4
Informational	4

5 Findings

5.1 Medium risk

5.1.1 Auction::_updateAuction - Incorrectly checking volt balance instead of vyper balance

Severity: *Medium risk*

Context: Auction.sol#L317

Description: After 8 days, the Vyper that gets minted is based of `VyperTreasury`.

```
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) : treasury.emitForAuction();

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

Before emitting the tokens from there, we do a check to make sure the `treasury` has enough tokens, but we incorrectly use `volt`, instead of `vyper`.

```
if (daySinceStart > 8 && volt.balanceOf(address(treasury)) == 0) revert
    TreasuryVoltIsEmpty();
```

The treasury only gets transferred Vyper to it, not Volt, we can confirm that in `VoltVyperNexus::swapVoltToVyperAndDistribute`

```
vyper.transfer(GENESIS_WALLET, wmul(vyperAmount, uint256(0.03e18)));
        vyper.transfer(LIQUIDITY_BONDING_ADDR, wmul(vyperAmount, uint256(0.07e18)));
        -> vyper.transfer(address(vyper.treasury()), wmul(vyperAmount, uint256(0.2
            e18)));
```

Currently, after day 8 the contract will start reverting on each `deposit`, since `_updateAuction` will revert.

Recommendation: Use `vyper.balanceOf` instead of `volt.balanceOf`.

Resolution: Fixed

5.1.2 `VoltVyperNexus::swapVoltAndDistribute` - Breaching the swapCap will brick some funds inside the contracts. Issue from previous audit persists

Severity: *Medium risk*

Context: VoltVyperNexus.sol#L133-L134

Description: One of the issues from the previous audit persists in two out of the three affected files.

Issue name: 6.3.2 Breaching the swapCap will brick some funds inside the contracts.

In `VyperDragonXNexus` the issue is fixed.

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

            //@note - Add the difference for the next day
            toDistribute += difference;

            currInterval.amountAllocated = swapCap;
        }
```

But in the other 2 Nexus' it isn't.

`VoltVyperNexus`

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

            //@note - Add the difference for the next day
            toDistribute += difference;

            currInterval.amountAllocated = swapCap;
        }

        //@issue M - M-02 from prev report persists
        _updateSnapshot();
```

`DragonXVoltNexus`

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

            //@note - Add the difference for the next day
            toDistribute += difference;

            currInterval.amountAllocated = swapCap;
        }

        //@issue M - M-02 from prev report persists
        _updateSnapshot();
```

Recommendation: Fix the issue in `DragonXVoltNexus` and `VoltVyperNexus`

Resolution: -

5.2 Low risk

5.2.1 `SwapActions.sol::swapExactInput` - There is no `deadline` param

Severity: *Low risk*

Context: SwapActions.sol#L156

Description: Inside `swapExactInput` the `deadline` param is missing, making the whole protocol unusable as the `checkDeadline` will revert.

```
ISwapRouter.ExactInputParams memory params = ISwapRouter.ExactInputParams({
            path: path,
            recipient: address(this),
            // deadline: deadline,
            amountIn: tokenInAmount,
            amountOutMinimum: minAmount
        });
```

```
    modifier checkDeadline(uint256 deadline) {
        require(_blockTimestamp() <= deadline, 'Transaction too old');
        _;
    }
```

Recommendation: Uncomment the `deadline` param

Resolution: Will be fixed when deploying on Mainnet

5.2.2 `_updateSnapshot` in all nexuses should check `lastSnapshot + 48 hours <= Time.blockTs()`

Severity: *Low risk*

Context: DragonXVoltNexus.sol#L275

Description:

Currently the check is as follows:

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

The issue here is that if exactly 48 hours pass from the last snapshot, we don't enter the `if` and we don't reset `toDistribute`, but the entire `toDistribute` has already been burned because of the `_intervalsForNewDay` and `_totalAmountForInterval` in `_calculateIntervals`

```
int128 _intervalsForNewDay = missedIntervals >= accumulatedIntervalsForTheDay
                ? (missedIntervals - accumulatedIntervalsForTheDay) + 1
                : 0;
            // console2.log("_intervalsForNewDay: ", _intervalsForNewDay);
            _totalAmountForInterval += (_intervalsForNewDay > INTERVALS_PER_DAY)
                ? uint128(toDistribute)
                : uint128(toDistribute / INTERVALS_PER_DAY) * _intervalsForNewDay;
```

We've already added the entire `toDistribute` to `_totalAmountForInterval`, since 48 have passed so we have to burn the entire `toDistribute`.

Because we don't reset it, we will re-add `toDistribute` to the `totalVoltDistributed`, but we've already burned it, so there is nothing left, but the contract thinks there is something. Now if new tokens enter the contract through the distribute function, they will start getting burned immediatly, but they should be burned the next day.

Test

The chances of this happening are very low, but it's worth fixing as it will break the distribution mechanism, as it will start distributing not for the following day, but for the current day as the code constantly thinks it has tokens it needs to distribute.

This happens in all nexus contracts.

Recommendation Change the check to:

```
lastSnapshot + 48 hours <= Time.blockTs()
```

Resolution: Fixed

5.2.3 `Auction#batchClaimableAmount` doesn't check if the day is today

Severity: *Low risk*

Context: Auction.sol#L166-L170

Description: Claiming is allowed only for passed days and there is a check inside `claim` to ensure that.

```
        if (_day == daySinceStart) revert OnlyClaimableTheNextDay();
```

However, that is no such check in `batchClaimableAmount` so it can return a higher amount than what the user can actually claim.

```
    function batchClaimableAmount(address _user, uint32[] calldata _days) public
        view returns (uint256 toClaim) {
        for (uint256 i; i < _days.length; ++i) {
            toClaim += amountToClaim(_user, _days[i]);
        }
    }
```

Resolution: Fixed

5.2.4 `Auction::batchClaimableAmount` - Can pass duplicate `_days`

Severity: *Low risk*

Context: Auction.sol#L166-L170

Description: When calling `batchClaimableAmount`, the caller can pass duplicate `_days`, so he can manipulate `toClaim` to whatever he wants basically.

```
//@issue I/L - can pass duplicate '_days'
    function batchClaimableAmount(address _user, uint32[] calldata _days) public
        view returns (uint256 toClaim) {
        for (uint256 i; i < _days.length; ++i) {
            toClaim += amountToClaim(_user, _days[i]);
        }
    }
```

Resolution: Acknowledged

5.3 Informational

5.3.1 Comment in `_updateAuction` is incorrect.

Severity: *Informational*

Context: Auction.sol#L311

Description `///@notice Emits the needed VOLT`

The function mints Vyper, not VOLT.

Resolution: Fixed

5.3.2 In `Auction::collectFees`, use `vyper.safeTransfer` instead of `transfer`, in order to keep the code consistent, as for `volt` and `dragonX` we use `safeTransfer`.

Severity: *Informational*

Context: Auction.sol#L221-L222

Resolution: Fixed

5.3.3 Change `_totalVoltDistributed` var name in `DragonXVoltNexus::_calculateIntervals`

Severity: *Informational*

Context: DragonXVoltNexus.sol#L226

Description In `DragonXVoltNexus::_calculateIntervals` there is a `_totalVoltDistributed` variable that should be renamed to `_totalDragonXDistributed` as the contract distributes DragonX tokens.

Resolution: Fixed

5.3.4 Only in `DragonXVoltInput.sol` swap cap set to 0 means `max.uint256`

Severity: *Informational*

Context: DragonXVoltInput.sol#L105-L106

Resolution: Fixed

Description In `DragonXVoltInput.sol` if 0 is passed for the swapCap, it will be set to `max.uint256`, however in the other `swapCap` functions it can be 0.