EGIS SECURITY

# SECURITY REVIEW FOR VOLT

FINDINGS SUMMARY
1 HIGH
7 LOW
1 INFO

DATES
13.09.2024 - 18.09.2024

## 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.

Both members of Egis Security have a proven track record on public auditing platforms such as Code4rena, Sherlock & Codehawks, uncovering more than 150 High/Medium severity vulnerabilities, with >1$70,000 in winnings and multiple solo/team 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 | Volt |
| Repository | Private |
| Commit hash | Private |
| Resolution | Private |
| Documentation | https://docs.volt.win/ |
| Methods | Manual review |

### Scope

| |
|---|
| /src/** |

### Issues Found

| | |
|---|---|
| Critical risk | 0 |
| High risk | 1 |
| Medium risk | 0 |
| Low risk | 7 |
| Informational | 1 |

# 5 Findings

## 5.1 High risk

### 5.1.1 alreadyAllocated is wrongly increased by whole dailyAllocation for the past day on each new snapshot

**Severity:** *High risk*

**Context:** VoltBuyAndBurn.sol#L379-L380

**Description:** Inside _calculateIntervals we have a branch, which is calculating missed intervals, which spans for multiple days. Because system requires to have a daily allocation, which is updated in the beginning of each new snapshot (24 hours), we have beforeCurrDay variable, which holds the amount of tokens that we have burned for missed intervals. It is then passed to _updateSnapshot as deltaAmount to calculate the new totalTitanXDistributed for the current day:

```
    function _updateSnapshot(uint256 deltaAmount) internal { // @ok
        if (Time.blockTs() < startTimeStamp || lastSnapshot + 24 hours > Time.
            blockTs()) return;

        uint32 timeElapsed = Time.blockTs() - startTimeStamp;

        uint32 snapshots = timeElapsed / 24 hours;

        uint256 balance = titanX.balanceOf(address(this));

        totalTitanXDistributed = deltaAmount > balance ? 0 : balance - deltaAmount;
        lastSnapshot = startTimeStamp + (snapshots * 24 hours);
    }
```

The problem is that on each new snapshot, we will assign wrong value to totalTitanXDistributed, because we wrongly assign deltaAmount to a value higher than the actual one. We always assign it to dailyAllocation, which will hold the allocation for the whole day. The correct way is to assign it to the value corresponding to the amount for the missing intervals for that day.

**Imagine the following:** For simplicity we assume daily allocation is 100%

1. On first day (snapshot) we have totalTitanXDistributed = 10_000
2. This means that we will have 288 intervals burning 34 tokens (10_000 / 288)
3. During this day distributeTitanXForBurning is called and we accrued additional 10_000 tokens, which should be set to totalTitanXDistributed for the next snapshot
4. We have burned 9_758 tokens for the first snapshot (we have only one more period to burn for), so the current contract balance is 10_000 + 242 = 10_242
5. We enter the next snapshot and call swapTitanXForVoltAndBurn at the middle of the day:

- inside _calculateIntervals we enter the **else** branch and for the first snapshot we correctly calculate interval amount of 34 tokens and add it to the _totalAmountForInterval, but what happens with alreadyAllocated is that we add the whole totalTitanXDistributed for the first snapshot, when we have only used 1/288 of it:

```
                uint256 forAllocation = Time.dayCountByT(
                    lastBurnedIntervalStartTimestamp) == dayOfLastInterval
```

```
                  ? totalTitanXDistributed
                  : balanceOf > alreadyAllocated + wmul(diff,
                     getDailyTitanXAllocation(end)) ? diff : 0;
              uint256 dailyAllocation = wmul(forAllocation,
                  getDailyTitanXAllocation(end));
              alreadyAllocated += dailyAllocation;
```

- On the next cycle iteration, we are supposed to add `5K` to `_totalAmountForInterval` (because we are calling the func at the middle of the snapshot), but we don't accrued that, because of the check:

```
    uint256 forAllocation = Time.dayCountByT(lastBurnedIntervalStartTimestamp) ==
        dayOfLastInterval ? totalTitanXDistributed
                  : balanceOf > alreadyAllocated + wmul(diff,
                     getDailyTitanXAllocation(end)) ? diff : 0;
```

1. We exit the func with wrong `beforeCurrDay` of 10K
2. We wrongly set `totalTitanXDistributed` to `10_242 - 10_000 = 242`, when it should be 10_000

**Recommendation:** Update `alreadyAllocated = _amountPerInterval * accumulatedIntervalsForTheDay`

**Resolution:** Fixed

## 5.2 Low risk

### 5.2.1 `forAllocation` calculation in `_calculateIntervals` else branch should be able to use whole contract balance as valid amount

**Severity:** *Low risk*

**Context:** VoltBuyAndBurn.sol#L363

**Description:** When we calculate the daily allocation for the current day we should consider that using the whole contract balance as valid allocation:

```
        uint256 diff = balanceOf > alreadyAllocate balanceOf - alreadyAllocated :
            0;

        //@note - If the day we are looping over the same  as the last interval's
            use the cached allocati otherwise use the current balance
        uint256 forAllocation = Time.dayCoun (lastBurnedIntervalStartTimestamp) ==
             dayOfLastInterval
            ? totalTitanXDistributed
            : balanceOf > alreadyAllocated + wmul(diff, getDailyTitanXAllocation(
                end)) ? diff : 0;
```

**Recommendation:** Use >= instead of > on following line:

```
balanceOf > alreadyAllocated + wmul(diff, getDailyTitanXAllocation(end)) ? diff : 0;
```

**Resolution:** Fixed

### 5.2.2 `getCurrentInterval` reverts if `startTimeStamp` is in the future

**Severity:** *Low risk*

**Context:** VoltBuyAndBurn.sol#L164-L165

**Description:** If someone calls `getCurrentInterval` before reaching `startTimeStamp`, function will revert with underflow.

**Recommendation:** Consider handling the case gracefully by either:

- returning zeros
- reverting with detailed error

**Resolution:** Acknowledged

### 5.2.3 Consider emitting events on important state changes in `VoltBuyAndBurn`

**Severity:** *Low risk*

**Context:** VoltBuyAndBurn.sol

**Description:**
Consider emitting events when: - `slippageAdmin` is changed - `titanXToVoltSlippage` is changed - `swapCap` is changed

**Resolution:** Acknowledged

### 5.2.4 First time we call `swapTitanXForVoldAndBurn` may result in `missedIntervals > INTERVALS_PER_DAY`

**Severity:** *Low risk*

**Context:** VoltBuyAndBurn.sol#L333-L335

**Description:** First time we calculate missed intervals we always use current day as `dayOfLastInterval`. That means that if intervals still haven't been updated and `startTimeStamp` is more than 24 hours in the past, we will use the allocation for the currnet day when we update the intevals to calculate the amount per inteval. The probability is very low, because when `distributeTitanXForBurning` we will try to update the intervals.

**Recommendation:** Consider assigning the day of `startTimeStamp` to `dayOfLastInterval`, if current day is > the day of `startTimeStamp`.

**Resolution:** Acknowledged

### 5.2.5 Division before multiplication in `VoltBuyAndBurn#_calculateIntervals`

**Severity:** *Low risk*

**Context:** VoltBuyAndBurn.sol#L339

**Description:**

```
            uint256 dailyAllocation = wmul(totalTitanXDistributed,
                getDailyTitanXAllocation(Time.blockTs()));
            uint128 _amountPerInterval = uint128(dailyAllocation / INTERVALS_PER_DAY
                );
            uint128 additionalAmount = _amountPerInterval * missedIntervals;
            _totalAmountForInterval = _amountPerInterval + additionalAmount;
```

Results in `_totalAmountForInterval` being lower than `alreadyAllocated`

**Resolution:** Acknowledged

### 5.2.6 Passing duplicate IDs will return incorrect `toClaim`

**Severity:** *Low risk*

**Context:** VoltAuction.sol#L138-L139

**Description:** Currently there is nothing stopping someone from calling `batchClaimableAmount` and passing duplicate id's in the `_ids` array, if duplicate ids are used then each "claim" will be counted twice or more. If an external integrator or front-end relies on the value returned from the function he can get tricked. Currently this function isn't used anywhere else in the protocol, thus keeping it Low.

**Recommendation:** Disallow duplicate entries in the `_ids` array.

**Resolution:** Acknowledged

### 5.2.7 If VaultAuction#amountToClaim is used by integrating parties, it may return wrong data for the current day

**Severity:** *Low risk*

**Context:** VoltAuction.sol#L150-L151

**Description:** Because `stats.titanXDeposited` will most probably change before the end of the day and the value returned from amountToClaim cannot be trusted if the id is for the current day

**Recommendation:** Consider reverting if the day is still pending, or documenting the behaviour (It cannot be trusted for the current day)

**Resolution:** Acknowledged

## 5.3 Informational

### 5.3.1 `VoltBuyAndBurn::swapTitanXForVoltAndBurn` - If swapCap is hit, the tokens will be distributed the next day

**Severity:** *Info risk*

**Context:** VoltBuyAndBurn.sol#L185

**Description:**
If we have missed to call a lot of intervals and we update them, we may have large `amountAllocated`, which will be limited to the `swapCap`. That results in leaving the tokens for the next snapshot (may be after 23 hours). In such situation we may not be burning tokens for current's snapshot intervals left, which is possible to do within the limit for every swap.

**Resolution:** Acknowledged