shieldify

Vyper Boost

SECURITY REVIEW

Date: 18 December 2024

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Vyper Boost

Powering a Lightning-Fast deflation with Electrifying buy and burn.

Volt, built on TitanX, is a hyper-deflationary token with a unique auction system. It features a capped supply with all tokens distributed in the first 10 days, triggering full deflation afterward. Volt utilizes 80% of system value to buy tokens + 8% of value for growing bonded liquidity growth. Volt enters deflation quickly with a massive buy and burn.

Learn more about Volt's concept and the technicalities behind it here.

4. Risk Classification

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

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors

- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 6 days with a total of 192 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified one High-severity issue, where 50% of the funds were not being sent to the treasury as intended. Additionally, a missing check in the updateAuction functionality could lead to a Denial of Service. The report also highlighted discrepancies between the implementation and the documentation, along with several other recommendations.

The Vyper development team has excelled in protocol development, testing, and their security approach, including conducting multiple security reviews.

5.1 Protocol Summary

Project Name	**Vyper Boost**
Repository	vyper-boost-contracts
Type of Project	DeFi, Staking
Audit Timeline	6 days
Review Commit Hash	005554b17b2337932cd473603562ef4d4b99606a
Fixes Review Commit Hash	ac39d48ff7593189398b7a91a108c42efeb4f9eb

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/VoltBurn.sol	52
src/Treasury.sol	28
src/BuyAndBurn.sol	161
src/Auction.sol	101
src/libs/PoolAddress.sol	30
src/libs/OracleLibrary.sol	46
src/interfaces/IVyper.sol	4
src/const/Constants.sol	20
src/actions/SwapActions.sol	83
Total	**525**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **High** issues: **1**
- **Low** issues: **2**
- **Informational** issues: **3**

ID	Title	Severity	Status
[H-01]	Vyper is Not Sent To The Treasury In `BuyAndBurn.swapDragonXToVyperAndBurn()` Function	High	Fixed
[L-01]	First Instance of Auction May Be DoS'd If `StartTimestamp` is Extremely Close to `Block.timestamp`	Low	Acknowledged
[L-02]	Constructor in `VoltBurn.sol` Does Not Check For Zero Address	Low	Fixed
[I-01]	Distribution of `DragonX` Is Different From `Docs` and `Code`	Informational	Fixed
[I-02]	`BuyAndBurn.swapDragonXToVyperAndBurn()` Function May Fail If Not Called For `>1` Day With Default Slippage	Informational	Acknowledged
[I-03]	Addresses Cannot Be Changed in The Protocol in Case of a Compromise	Informational	Acknowledged

7. Findings

[H-01] Vyper is Not Sent To The Treasury In `BuyAndBurn.swapDragonXToVyperAndBurn()` Function

Severity

High Risk

Description

In `BuyAndBurn.swapDragonXToVyperAndBurn()` function, once DragonX is swapped to Vyper, 8% goes to `LIQUIDITY_BONDING_ADDR` and the rest is burned. Note that the 50% fund transfer to `vyper.treasury()` is commented out.

Location of Affected Code

File: src/BuyAndBurn.sol#L116

```
function swapDragonXToVyperAndBurn(uint32 _deadline) external
   intervalUpdate notExpired(_deadline) {
    // code
    uint256 incentive = wmul(currInterval.amountAllocated, INCENTIVE_FEE)
       ;

    uint256 dragonXToSwapAndBurn = currInterval.amountAllocated -
       incentive;

    uint256 vyperAmount = swapExactInput(address(dragonX), address(vyper)
       , dragonXToSwapAndBurn, 0, _deadline);

    {
    ///@note - Allocations
    @>  vyper.transfer(LIQUIDITY_BONDING_ADDR, wmul(vyperAmount, uint256
       (0.08e18)));
    // vyper.transfer(address(vyper.treasury()), wmul(vyperAmount,
       uint256(0.5e18)));
    @>  burnVyper();
    }

    dragonX.safeTransfer(msg.sender, incentive);
    // code
}
```

Impact

Too many funds are burned, the treasury will not get the funds.

Recommendation

Uncomment the code to allow transfers to the Vyper treasury as well.

Team Response

Fixed.

[L-01] First Instance of Auction May Be DoS'd If `StartTimestamp` is Extremely Close to `Block.timestamp`

Severity

Low Risk

Description

In `Auction.sol`, users can call `deposit()` and deposit dragonX and get Vyper back.

When `deposit()` is first called, `_updateAuction()` will be called which sets `dailyStats[daySinceStart].vyperEmitted`. The `_updateAuction()` function also checks if the Vyper balance in the treasury is empty.

Location of Affected Code

File: src//Auction.sol#L135

```
    ///@notice Emits the needed VYPER
    function _updateAuction() internal {
    uint32 daySinceStart = Time.dayGap(startTimestamp, Time.blockTs()) +
       1;

    if (dailyStats[daySinceStart].vyperEmitted != 0) return;

    if (vyper.balanceOf(address(treasury)) == 0) revert
       TreasuryVoltIsEmpty();

    uint256 emitted = treasury.emitForAuction();

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

Once `_updateAuction()` is called, it will not be called again until the next day (`86400` seconds passes).

In the test files, 500M tokens is dealt in the Treasury, and in the first call to deposit, 100M tokens (20%) are sent to the Auction.

However, this may not be the case in reality since there can be a lag time between the creation of Auction.sol, the depositing of Vyper tokens, and the calling of `deposit()`

An edge case can happen where if the `startTimeStamp` is extremely close to the `block.timestamp` , and the Auction.sol contract is created.

A user can deposit 1 wei Vyper token directly into the treasury and call `deposit()` , frontrunning the protocol's deposit of Vyper tokens in the treasury. This way, he will DoS the first day of the Auction, which will affect people since they might not know that they are eligible for zero Vyper tokens.

Impact

On the first day of the auction may distribute zero Vyper tokens to many dragonX depositors, making the dragonX depositors lose out.

Recommendation

In the constructor, it will be good to check that the `startTimestamp` is greater than the `block.timestamp` , so that the Auction will only start in the future.

Additionally, add more checks in the deposit function, ensuring that only if the Treasury has a certain amount of Vyper (instead of checking for zero) before commencing the Auction.

Team Response

Acknowledged! We usually deposit 2-3 hours prior to the startTimestamp, so that we have time to do everything.

[L-02] Constructor in `VoltBurn.sol` Does Not Check For Zero Address

Severity

Low Risk

Description

In the constructor of `VoltBurn.sol`, there is no check for zero address.

```
constructor(address _dragonX, address _volt, SwapActionParams memory _s)
    SwapActions(_s) {
     dragonX = ERC20Burnable(_dragonX);
     //code
}
```

The other contracts use the `notAddress0` modifier to check for zero address.

```
constructor(uint32 _startTimestamp, address _vyper, address _dragonX,
    VyperBoostBuyAndBurn _bnb, address _voltBurn)
     notAddress0(_vyper)
     notAddress0(_dragonX)
     notAddress0(_voltBurn)
     notAddress0(address(_bnb))
{
```

Impact

Lack of sanity check may result in redeployment and wasted gas.

Recommendation

Recommend adding `notAddress0(x)` in the `VoltBurn.sol` constructor as well and check for the address of `_dragonX` and `_volt`.

Team Response

Fixed.

[I-01] Distribution of `DragonX` Is Different From `Docs` and `Code`

Severity

Informational

Description

In the docs, it mentions:

```
Distribution of DragonX input
- 7% Genesis
- 1% Dev
- 4% Volt burn
- 4% burnt
- 8% liquidity wallet
- 76% buy and burn
```

The coded distribution is slightly different:

```
uint64 constant TO_GENESIS = 0.07e18; //7%
uint64 constant TO_DEV_WALLET = 0.01e18; //1%
uint64 constant DX_BURN = 0.05e18; //5%
uint64 constant TO_LP = 0.1e18; //10%
uint64 constant TO_VOLT_BURN = 0.05e18; //5%
uint64 constant TO_BNB = 0.72e18; //72%

Distribution of DragonX input
- 7% Genesis
- 1% Dev
- 5% Volt burn *
- 5% burnt *
- 10% liquidity wallet *
- 72% buy and burn *
```

Impact

Discrepancy in docs and code.

Recommendation

Although it still adds up to 100%, if the code is the latest, we recommend updating the docs.

Team Response

Fixed.

[I-02] `BuyAndBurn.swapDragonXToVyperAndBurn()` Function May Fail If Not Called For `>1` Day With Default Slippage

Severity

Informational

Description

If slippage is not set, the default slippage when swapping DragonX to Vyper is `1-0.2e18 = 0.8e18`
.

```
function getTwapAmount(address tokenIn, address tokenOut, uint256 amount)
    public
    view
    returns (uint256 twapAmount, uint224 slippage)
{
    // code

    if (slippageConfig.twapLookback == 0 && slippageConfig.slippage == 0)
        {
@>      slippageConfig = Slippage({twapLookback: 15, slippage: WAD - 0.2
   e18});
    }

    // code
}
```

If slippage is not set and `swapDragonXToVyperAndBurn()` is not called for `>24` hours (The last time it was `~23` hours), the swap will fail until the owner changes the config.

PoC:

```
function test_4() public prepareBnB {
    vm.prank(bnb.owner());
@>  // No slippage is set, default becomes 0.8e18
@>  // bnb.changeSlippageConfig(0
   x214CAD3f7FbBe66919968Fa3a1b16E84cFcd457F, 0, 15);
    console.log("---------------------------------");
    console.log("Burner Balance DragonX:", dragonX.balanceOf(address(
       burner)));
    console.log("Balance DragonX:", dragonX.balanceOf(address(bnb)));
    console.log("DXD", bnb.totalDragonXDistributed());
    console.log("Snap", bnb.lastSnapshot());
    console.log("Interval:", bnb.lastIntervalNumber());
    console.log("Total DragonX Distributed", bnb.totalDragonXDistributed
    ());
```

```
    // 24 hours later
@>  vm.warp(bnb.startTimeStamp() + 24 hours );
    vm.prank(burner,burner);
    bnb.swapDragonXToVyperAndBurn(type(uint32).max);
    console.log("-----------------------------------");
    console.log("Burner Balance DragonX:", dragonX.balanceOf(address(
       burner)));
    console.log("Balance DragonX:", dragonX.balanceOf(address(bnb)));
    console.log("DXD", bnb.totalDragonXDistributed());
    console.log("Snap", bnb.lastSnapshot());
    console.log("Interval:", bnb.lastIntervalNumber());
    console.log("Total DragonX Distributed", bnb.totalDragonXDistributed
    ());
}
```

The test will fail with:

```
Failing tests:
Encountered 1 failing test in test/unit/BuyAndBurnTest.sol:BuyAndBurnTest
[FAIL: revert: Too little received] test_4() (gas: 763361)
```

The issue is just for the protocol's information. Note that the owner can simply change the slippage, but if the owner is unaware that the function has not been called for a long time, then many users who try to call the function will find the transaction reverting.

Impact

Users will waste gas if they call and the transaction reverts.

Recommendation

Take note if the function `swapDragonXToVyperAndBurn()` has not been called for a long time. This can be the case in the far future, when there's not much incoming DragonX into the contract, and users are not incentivized to call burn until enough dragonX is accumulated in the contract because they will spend more in gas .

Slippage should be adjusted accordingly.

Team Response

Acknowledged.

[I-03] Addresses Cannot Be Changed in The Protocol in Case of a Compromise

Severity

Informational

Description

In `BuyAndBurn.swapDragonXToVyperAndBurn()` function, the `LIQUIDITY_BONDING_ADDR` that receives \codex{8%} of the Vyper amount is set directly.

```
vyper.transfer(LIQUIDITY_BONDING_ADDR, wmul(vyperAmount, uint256(0.08e18)
    ));
```

In **Constants.sol**:

```
address constant LIQUIDITY_BONDING_ADDR = 0
    x45C03d66229d01dF2645E813222b16C8B8b86894;
```

If the `LIQUIDITY_BONDING_ADDR` is compromised, the owner cannot redirect funds to another wallet address.

Same goes to all the fixed addresses in `Auction._distribute()`.

```
function _distribute(uint256 _amount) internal {
    dragonX.transfer(DEAD_ADDR, wmul(_amount, DX_BURN));

@>  dragonX.transfer(LIQUIDITY_BONDING_ADDR, wmul(_amount, TO_LP));
@>  dragonX.transfer(DEV_WALLET, wmul(_amount, TO_DEV_WALLET));
@>  dragonX.transfer(GENESIS_WALLET, wmul(_amount, TO_GENESIS));
    dragonX.transfer(voltBurn, wmul(_amount, TO_VOLT_BURN));

    // code
}
```

Impact

The wallet can be compromised.

Recommendation

This issue is information for the protocol to acknowledge since all previous on-chain contracts have this fixed address pattern. Should changes be made, best practice is to allow the owner to change the address with an access-controlled function.

Team Response

Acknowledged.

shieldify

Thank you!