



**Phage**  
SECURITY

# DFDX

Security Review

Conducted by:

**Pyro**, Security Researcher

## Table of Contents

<b>Disclaimer</b>	<b>3</b>
<b>System overview</b>	<b>3</b>
<b>Executive summary</b>	<b>3</b>
Overview . . . . .	3
Timeline . . . . .	3
Scope . . . . .	4
Issues Found . . . . .	4
<b>Findings</b>	<b>5</b>
Medium Severity . . . . .	5
[M-01] Contract can be DOSed . . . . .	5
Low Severity . . . . .	5
[L-01] V3 pools can be skewed . . . . .	5
[L-02] IUniswapV3Pool.initialize can be called by an outside entity to manipulate the price . . . . .	6
[L-03] _encodeSqrtRatioX96 may overflow . . . . .	7
[L-04] If a migrator is ever set the LP can be unlocked instantly . . . . .	8
[L-05] Pools can be MEVed . . . . .	8



## 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**.

## System overview

**DFDX** is a meme token on TitanX. Its main contract sets up the token and deploys both Uniswap v2 and v3 pools.

## Executive summary

### Overview

Project Name	DFDX
Repository	private
Commit hash	0da4e0acd24fc03e059b3e3497dfbc0569de2363
Remediation	1996c4a548aaeea0911692b4884b6845d7d474ec
Methods	Manual review

### Timeline

Audit kick-off	17.01.2025
End of audit	19.01.2025
Remediations start	20.01.2025

## Scope

src/DFDX.sol

## Issues Found

Severity	Count
High	0
Medium	1
Low	5

## Findings

### Medium Severity

#### [M-01] Contract can be DOSed

During initialize we add liquidity to UNIV2.

```
uniswapV2Router.addLiquidity(
    address(this),
    WETH,
    INITIAL_LP_DFDX,
    INITIAL_LP_WETH,
    INITIAL_LP_DFDX,
    INITIAL_LP_WETH,
    address(this),
    deadline
);
```

However a malicious actor can create a pool before us, send some WETH and call sync, which would change the ratio of the pair. This would be problematic as uniswapV2Router.addLiquidity would expect 100% of the tokens to be utilized, however since the pair is skewed one of the tokens would be under-utilized and because of it addLiquidity would revert.

#### Recommendation:

Check if the other token balance > 0. If so, calculate how much DFDX should be sent to the pair to make the ratio correct. Send it to the pair and call sync on the pool. Then call addLiquidity, but make sure there is a small slippage tolerance available or else if the ratio is not perfect the TX would still revert.

#### Resolution: Fixed

---

### Low Severity

#### [L-01] V3 pools can be skewed

DFDX creates UNIV3 pools inside it's constructor:

```

DFDX_DX_V3_PAIR = v3Factory.createPool(address(this), _DX, V3_POOL_FEE);
DFDX_WETH_V3_PAIR = v3Factory.createPool(
    address(this),
    WETH,
    V3_POOL_FEE
);

// But adds liquidity in initialize (uniswapV3PositionManager.mint):
uniswapV3PositionManager.mint(
    INonfungiblePositionManager.MintParams({
        token0: _token0,
        token1: _token1,
        fee: V3_POOL_FEE,
        tickLower: (MIN_TICK / tickSpacing) * tickSpacing,
        tickUpper: (MAX_TICK / tickSpacing) * tickSpacing,
        amount0Desired: _amount0Desired,
        amount1Desired: _amount1Desired,
        amount0Min: 0,
        amount1Min: 0,
        recipient: msg.sender,
        deadline: deadline
    })
);
  
```

That combined with the fact that we have 0 as amountOut can provide an opportunity for a malicious actor to manipulate the price. This is possible, as we can supply 0 liquidity to an empty V3 pool and move the ticks up to min/max tick - <https://x.com/0xKaden/status/1856784539978444827>

#### **Recommendation:**

Either make sure both contract creation and initialization are in 1 TX, or set appropriate amountOut values.

#### **Resolution:** Fixed

### [L-02] IUniswapV3Pool.initialize can be called by an outside entity to manipulate the price

The contract first creates both UNI pools inside the constructor:

```

DFDX_DX_V3_PAIR = v3Factory.createPool(address(this), _DX, V3_POOL_FEE);
DFDX_WETH_V3_PAIR = v3Factory.createPool(
    address(this),
    WETH,
    V3_POOL_FEE
);
  
```

Then it initializes them with sqrtPriceX96DX:



```
IUniswapV3Pool(DFDX_DX_V3_PAIR).initialize(sqrtPriceX96DX);
```

However a malicious actor can send a TX to the UNIV3 pool, between contract creation and initialized, in order to initialize the pool with his preferred X96 value. This of course will cause the price between those assets to be different than the one intended by the system, enabling him to exploit the difference if any liquidity is added.

#### **Recommendation:**

Either add initialize operations to the constructor or create and initialize the contract in one TX.

#### **Resolution:** Fixed

### [L-03] \_encodeSqrtRatioX96 may overflow

Currently when we use `_encodeSqrtRatioX96` for calculating the X96 ratio for the UNI pools:

```
function _encodeSqrtRatioX96(
    uint256 amount0,
    uint256 amount1
) internal pure returns (uint160) {
    uint160 sqrtPX96 = uint160(
        (Math.sqrt((amount1 * PRECISION) / amount0) << 96) / PRECISION_SQRT
    );
    return sqrtPX96;
}
```

When we do so, we use `PRECISION`, which is `1e48` to make sure we keep the value more precise. However when doing so during the multiplication between `amount1 * PRECISION` we get really close to the `uint256` maximum. Currently if `INITIAL_LP_DX` is `amount1`, the result from the multiplication would be  $1e26 * 1e48 = 1e74$  and the `uint256` max  $1.15e77$ .

If before deployment the values for any one of those amount is changed, this may result in the above math overflowing and bricking the contract.

#### **Recommendation:**

If the same amounts are kept and deployed with there won't be an issue, however if they plan to be changed make sure the new values will work with this math, or lower the precision.

#### **Resolution:** Acknowledged



#### [L-04] If a migrator is ever set the LP can be unlocked instantly

Currently UniswapV2Locker has a migrate function that allows for the user locking the tokens to unlock them if the migrator is set. This is done in order to allow users to migrate from UNI V2 to V3.

However it would currently allow the contract deployer to withdraw all of the LP. #####  
Recommendation:

The probability of UniswapV2Locker to go through a migration in the next year is gonna be low, however for 100% security consider implementing a small locking contract that would lock the LP tokens. This way the system would also save on fees.

**Resolution:** Acknowledged

#### [L-05] Pools can be MEVed

When creating pools we create 2 UNIv3. One DFDX : DX and one DFDX : WETH, where the initial LP for the pools would be 22mil DFDX : 100mil DX and 22mil DFDX : 0.3 WETH respectfully.

However a MEV opportunity may appear right after we create them, as these 2 pools would mean that we are artificially setting the dragonX : WETH pool ratio for this pair to 100mil : 0.3 WETH. That is because if open a path 100mil DX → DFDX → 0.3 WETH.

Currently the real price of 0.3 WETH is ~1100 USD, however 100mil in dragonX is only 413 USD, meaning that these 2 pools would be used by MEV bots to arbitrage the price to the real one dragonX pool and steal the profit.

**Recommendation:**

Consider before deployment to fetch the real price and make the pools according to it.

**Resolution:** Acknowledged