shieldify

DFDX

SECURITY REVIEW

Date: 21 January 2025

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, capable of auditing 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 DFDX

Dragon Fart Dust is a meme on Ethereum. Our vision is to tap into the fun and excitement that a well built and cohesive community can embrace in the current crypto meme culture. Although there is simplicity built into the code, and anyone can easily access with their ETH on Uniswap, we plan to build lore and light hearted announcements on our social accounts that will bring everyone together that chooses to participate. The Dragon is full of magic, very gassy and the only remedy is green candles.

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 2 days with a total of 64 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified Low and Informational severity issues related to lack of slippage protection by adding liquidity and lack of lock mechanism.

The Dfdx dev team has been very responsive to the Shieldify research team's inquiries, demonstrating a strong commitment and dedication to the protocol's development.

5.1 Protocol Summary

Project Name	**DFDX**
Repository	dfdx-erc20
Type of Project	ERC-20
Audit Timeline	2 days
Review Commit Hash	0da4e0acd24fc03e059b3e3497dfbc0569de2363
Fixes Review Commit Hash	1996c4a548aaeea0911692b4884b6845d7d474ec

5.2 Scope

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

File	nSLOC
src/DFDX.sol	244
Total	**244**

6. Findings Summary

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

- **Low** issues: **1**
- **Informational** issues: **1**

ID	Title	Severity	Status
[L-01]	The `amount0Min` and `amount1Min` Set to Zero in Uniswap V3 Liquidity Addition	Low	Fixed
[I-01]	Unsecured Uniswap V3 Liquidity Positions Due to Missing Lock Mechanism Allow Immediate Deployer Access	Informational	Acknowledged

7. Findings

[L-01] The `amount0Min` and `amount1Min` Set to Zero in Uniswap V3 Liquidity Addition

Severity

Low

Description

The `initialize()` function of the DFDX contract sets `amount0Min` and `amount1Min` to zero when adding liquidity to Uniswap V3 pools. This practice exposes the protocol to sandwich attacks, as it allows front-running transactions to manipulate the prices and potentially result in a loss of value during liquidity provision.

Setting these parameters to non-zero values can protect against price slippage and mitigate the risk of sandwich attacks. The Uniswap V3 documentation explicitly states that setting amount0Min and amount1Min to zero introduces a vulnerability in production.

Location of Affected Code

File: src/DFDX.sol#L212

```
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, // Vulnerable: should not be 0
        amount1Min: 0, // Vulnerable: should not be 0
        recipient: msg.sender,
        deadline: deadline
    })
);
```

Impact

A function calling mint with no slippage protection would be vulnerable to a frontrunning attack designed to execute the mint call at an inaccurate price.

Recommendation

Modify the `initialize()` function to compute reasonable non-zero values for `amount0Min` and `amount1Min`. These values should represent the minimum acceptable amounts based on the expected price range, ensuring that liquidity is only added when the pool's price remains within an acceptable range.

```
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: _amount0Desired * 98 / 100, // Example: Allow max 2%
           slippage
        amount1Min: _amount1Desired * 98 / 100, // Example: Allow max 2%
           slippage
        recipient: msg.sender,
        deadline: deadline
    })
);
```

Team Response

Fixed.

[I-01] Unsecured Uniswap V3 Liquidity Positions Due to Missing Lock Mechanism Allow Immediate Deployer Access

Severity

Informational

Description

For Uniswap V2, the liquidity provider (LP) tokens received after adding liquidity are explicitly locked using the `uniswapV2Locker.lockLPToken()` function. This ensures that the LP tokens cannot be transferred or used until the lock duration expires:

```
uniswapV2Locker.lockLPToken{value: lockerFee}(
    DFDX_DX_V2_PAIR,
    IERC20(DFDX_DX_V2_PAIR).balanceOf(address(this)),
    block.timestamp + LOCK_DURATION,
    payable(address(0)),
    true,
    payable(msg.sender)
);

uniswapV2Locker.lockLPToken{value: lockerFee}(
    DFDX_WETH_V2_PAIR,
    IERC20(DFDX_WETH_V2_PAIR).balanceOf(address(this)),
    block.timestamp + LOCK_DURATION,
    payable(address(0)),
    true,
    payable(msg.sender)
);
```

For Uniswap V3, the code mints new liquidity positions using the uniswapV3PositionManager.mint function. However, these positions are directly assigned to the `msg.sender`, and no mechanism locks them. This means the liquidity positions can be immediately utilized, transferred, or withdrawn by the deployer without any restrictions:

```
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, // Bug: Positions minted directly to msg.
           sender
        deadline: deadline
    })
);
```

So clearly there is an inconsistency after adding liquidity as in one case LP tokens are locked and in another case it is minted to `msg.sender` and not locked.

Location of Affected Code

File: src/DFDX.sol#L214

```
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, // Bug: Positions minted directly to msg.
           sender
        deadline: deadline
    })
);
```

Impact

The Uniswap V3 LP positions are not subject to the same lock mechanism as Uniswap V2 LP tokens, allowing the deployer to access them immediately. The deployer could withdraw or sell the V3 positions, undermining the intended security of the liquidity provided.

Recommendation

To resolve the issue, you should implement a locking mechanism for the Uniswap V3 positions.

Team Response

Acknowledged.

shieldify

Thank you!