Firefly Audit

February 1, 2024

My background as a smart contract auditor includes audits for Arrakis Finance, Nouns.WTF, and LooksRare.

1. Core-V2

As a result of my review, the contracts forked the `UniswapV2` codebase are almost the same except replacing the protocol name `Uniswap` as `Firefly`. Therefore, there is nothing to be considered specially.

2. Core-V3

As `Core-V2` contracts did, these `Core-V3` contracts are also forked from the `UniswapV3` codebase almost the same but they introduced the new functionality that can update the fee of the deployed pool.

Impact: Nobody can execute `changePairFee` function after deploying.

Severity: **High**

Vulnerability Details

```
    function changePairFee(
        address tokenA,
        address tokenB,
        uint24 currentFee,
        uint24 newFee,
        int24 newTickSpacing
    ) external override adminRequired {
        require(tokenA != address(0) && tokenB != address(0)); // Invalid token
addresses
        require(currentFee != newFee); // New fee must be different from current
fee
```

```
        require(newFee <= MAX_FEE); // New fee exceeds maximum allowed
        require(feeAmountTickSpacing[newFee] == newTickSpacing); // New fee or
tick spacing not enabled

        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) :
(tokenB, tokenA);
        address pool = getPool[token0][token1][currentFee];
        require(pool != address(0)); // Pool does not exist

        // Update mappings
        getPool[token0][token1][newFee] = pool;
        getPool[token1][token0][newFee] = pool;

        // Remove old fee mapping
        delete getPool[token0][token1][currentFee];
        delete getPool[token1][token0][currentFee];

        // Emit the PairFeeChanged event after updating the fee
        emit PairFeeChanged(tokenA, tokenB, currentFee, newFee, newTickSpacing);
    }
```

I think they constructed well the code including updating `getPool` mappings and removing old ones.

And this factory contract introduced `Admin.sol` contract for admin privilege of `changePairFee` function

However, I think there is an issue in this part. `Admin.sol` has no logic that initialize the first `admin`. And even `addAdmin`, `removeAdmin` functions have `adminRequired` modifier. This means that nobody can add/remove admins after deployed. And this also means that nobody can execute `changePairFee` function after deploying.

Recommended Mitigation

I think the logic that initializing first admin should be added.

```
contract Admins {
    /// @notice Mapping to keep track of addresses with admin privileges
    mapping(address => bool) public admins;

    /// @notice Modifier to restrict function access to only admins
    modifier adminRequired() {
        require(isAdmin(msg.sender)); // Caller is not an admin
        _;
    } // @audit there is no initializing first admin
```

```
    /// @notice Emitted when a new admin is added
    /// @param newAdmin The address of the newly added admin
    /// @param addedBy The address of the admin who added the new admin
    event AdminAdded(address indexed newAdmin, address indexed addedBy);

    /// @notice Emitted when an admin is removed
    /// @param oldAdmin The address of the removed admin
    /// @param removedBy The address of the admin who removed the old admin
    event AdminRemoved(address indexed oldAdmin, address indexed removedBy);

    /// @notice Checks if an address has admin privileges
    /// @param _checkOwner The address to check for admin privileges
    /// @return True if the address has admin privileges, false otherwise
    function isAdmin(address _checkOwner) public view returns(bool) {
        return admins[_checkOwner];
    }

    /// @notice Adds a new admin address
    /// @param _newAdmin The address to grant admin privileges
@>  function addAdmin(address _newAdmin) public adminRequired {
        require(_newAdmin != address(0)); // Invalid admin address
        require(!admins[_newAdmin]); // Address is already an admin

        admins[_newAdmin] = true;
        emit AdminAdded(_newAdmin, msg.sender);
    }

    /// @notice Removes an existing admin address
    /// @param _oldAdmin The address to revoke admin privileges from
@>  function removeAdmin(address _oldAdmin) public adminRequired {
        require(_oldAdmin != address(0)); // Invalid admin address
        require(admins[_oldAdmin]); // Address is not an admin

        admins[_oldAdmin] = false;
        emit AdminRemoved(_oldAdmin, msg.sender);
    }
}
```

3. lib

Forked entirely.

4. swap-router-contracts

Almost same. (Just replacing the `uniswap` as `firefly`)

5. universal-router

Almost same. (Just replacing the `uniswap` as `firefly`)

6. v2-periphery.contracts

Almost same. (Just replacing the `uniswap` as `firefly`)

7. v3-periphery.contracts

Almost same. (Just replacing the `uniswap` as `firefly`)

8. v3-staker.contracts

Almost same. (Just replacing the `uniswap` as `firefly`)

Personal Opinion for Forward Developing

Uniswap has several potential issues when it interacts with external DeFi or DeX protocol.

For example, pool.slot0 of UniswapV3Pool has potential risk to may occur price manipulation by sandwich attacking during swap. Of course, it depends on how constructed and develop the logic and codebase according to your roadmap and plan.

To avoid and fix these potential risks and possible issues in forward developing, should work with code by considering known potential vulnerabilities of Uniswap V3, and typical DEX issues and perform the pre/post-auditing code completely.