SHERLOCK

Sherlock Security Review For Telcoin

Public contest prepared for: **Telcoin**
Lead Security Expert: **0x73696d616f**
Date Audited: **November 6 - November 9, 2024**

Introduction

Telcoin leverages blockchain technology to provide access to low-cost, high-quality decentralized financial products for every mobile phone user in the world. This audit focuses on upgrading the swapping mechanism Telcoin uses to better accommodate Stablecoins.

Scope

Repository: telcoin/telcoin-audit

Branch: main

Audited Commit: 15c5381f16f6a7febd9a07cba2f1f77fbce2184f

Final Commit: 813c73668bb7d28e66a35e973db456aeb1e8598e

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
1	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0xjarix
BengalCatBalu
0x73696d616f
John44
jsmi
Abhan1041
newspacexyz
KiroBrejka
rsam_eth
rscodes
hals
0xNirix
dany.armstrong90
Bigsam
0xlucky
novaman33
y4y

Issue M-1: `AmirX.defiToStablecoinSwap()` doesn't verify the stable coin origin amount parameters after conducting the defi swap which might result in exceeding (violating) the burn limit of the origin XYZ stablecoin

Source: https://github.com/sherlock-audit/2024-11-telcoin-judging/issues/142

Found by

0x73696d616f, 0xNirix, 0xjarix, 0xlucky, Abhan1041, BengalCatBalu, Bigsam, John44, KiroBrejka, dany.armstrong90, hals, jsmi, newspacexyz, novaman33, rsam_eth, rscodes, y4y

Summary

`AmirX.defiToStablecoinSwap()` function checks the validity of the stablecoin swap parameters/amounts via `_verifyStablecoinSwap()` before calling `_defiSwap()` which will result in `Stablecoin(ss.origin).totalSupply()-ss.oAmount` exceeding the `minSupply` of that origin token if it's a stable XYZ token.

Root Cause

The root cause is checking the burn limit of the `ss.oAmount` (when verifying the stablecoinSwap parameters) **before** doing the external swap `_defiSwap()` that might return results different from the initial `ss.oAmount`.

Internal pre-conditions

- `AmirX.sol` contract interacts with three types of tokens : USDC, USDT & XYZ stable tokens that is identified by the protocol, where each of these XYZ tokens has a `maxSupply` & a `minSupply`:

```
struct eXYZ {
    // status of address as stablecoin
    bool validity;
    // the max mint limit
    uint256 maxSupply;
    // the min burn limit
```

```
    uint256 minSupply;
}
```

- The contract enables users from doing swaps in the following directions:

1. `defiToStablecoinSwap()` : if the user has `USDC` and wants to get `XYZ2`, the swap is done from `USDC` to `XYZ1` by an external aggregator via `_defiSwap()`, then `XYZ1` to `XYZ2` handeled internally via `_stablecoinSwap()`.
2. `stablecoinToDefiSwap()` : when the user has `XYZ1` token and wants to trade it for `USDC`, the swap is done first internally to swap `XYZ1` to `XYZ2` via `_stablecoinSwap()`, then the swapped `XYZ2` tokens are traded for `USDC` by an external aggregator via `_defiSwap()`.
3. `defiSwap()` : when the user has `USDC` and wants to trade it for `XYZ` token or any other token, and this is done by an external aggregator via `_defiSwap()`.
4. `stablecoinSwap()` : when the user has `XYZ1` and wants to trade it for `XYZ2` tokens.

note: `XYZ1` & `XYZ2` here are a recognized (registered) stablecoins by the `StablecoinHandler.sol` that are deployed by the protocol (`Stablecoin.sol`), and these tokens are referred as `ss.origin` & `ss.target` and can be any other ERC20 token approved by the protocol (USDC/USDT), where a `liquiditySafe` will be the intermediate address to transfer tokens from origin to target.

- If we looked at the (where the issue is):

```
function defiToStablecoinSwap(
      address wallet,
      StablecoinSwap memory ss,
      DefiSwap memory defi
   ) external payable onlyRole(SWAPPER_ROLE) whenNotPaused {
      // checks if defi will fail
      _verifyDefiSwap(wallet, defi);
      // checks if stablecoin swap will fail
      _verifyStablecoinSwap(wallet, ss);

      //check balance to adjust second swap
      uint256 iBalance = ERC20(ss.origin).balanceOf(wallet);
      _defiSwap(wallet, defi);
      uint256 fBalance = ERC20(ss.origin).balanceOf(wallet);
      ss.oAmount = fBalance - iBalance;
      //change balance to reflect change
      _stablecoinSwap(wallet, ss);
   }
```

1. first a check is made to verify the defiSwap parameters (via `_verifyDefiSwap()`) and then the stablecoinSwap parameters are checked (via `_verifyStablecoinSwap`):

```
function _verifyStablecoinSwap(
        address wallet,
```

```
        StablecoinSwap memory ss
    ) internal view nonZero(ss) {
        // Ensure the wallet address is not zero.
        if (wallet == address(0)) revert ZeroValueInput("WALLET");

        // For the origin currency:
        if (isXYZ(ss.origin)) {
            // Ensure the total supply does not drop below the minimum limit after
↪   burning the specified amount.
            if (
                Stablecoin(ss.origin).totalSupply() - ss.oAmount <
                getMinLimit(ss.origin)
            ) revert InvalidMintBurnBoundry(ss.origin, ss.oAmount);
        } else if (ss.liquiditySafe == address(0)) {
            // Ensure the liquidity safe is provided for ERC20 origin tokens.
            revert ZeroValueInput("LIQUIDITY SAFE");
        }

        // For the target currency:
        if (isXYZ(ss.target)) {
            // Ensure the total supply does not exceed the maximum limit after
↪   minting the specified amount.
            if (
                Stablecoin(ss.target).totalSupply() + ss.tAmount >
                getMaxLimit(ss.target)
            ) revert InvalidMintBurnBoundry(ss.target, ss.tAmount);
        } else if (ss.liquiditySafe == address(0)) {
            // Ensure the liquidity safe is provided for ERC20 target tokens.
            revert ZeroValueInput("LIQUIDITY SAFE");
        }
    }
```

as can be seen, if the origin token (`ss.origin`) is a stable coin `XYZ1`; a check is made on the origin `XYZ1` amount (`ss.oAmount` that is going to be burnt from the user) to not violate the `minSupply` after burning :

```
if (isXYZ(ss.origin)) {
            // Ensure the total supply does not drop below the minimum limit after
↪   burning the specified amount.
            if (
                Stablecoin(ss.origin).totalSupply() - ss.oAmount <
                getMinLimit(ss.origin)
            ) revert InvalidMintBurnBoundry(ss.origin, ss.oAmount);
        }
```

2. then the balance of the user wallet is cached before doing the external swap, and after the external swap that is done by the external aggregator, and if there's a difference; the `ss.oAmount` is updated accordingly, and the internal swap is done (via `_stablecoinSwap()`).

- The verification on the stablecoinSwap `ss.oAmount` parameter **is done before getting the actual amount of the `ss.oAmount` that is received when calling `_defiSwap()`**, so if the `_defiSwap()` call returns a balance difference of `XYZ1` (`ss.origin`) token different from the `ss.oAmount`; then the check made by the `_verifyStablecoinSwap()` to validate the amount of `XYZ1` token to be burnt might be violated if the actual modified `ss.oAmount` is greater than the `ss.oAmount` initially checked for.
- The same issue presents in the (`AmirX.swap()`)[https://github.com/sherlock-audit/2024-11-telcoin/blob/b9c751b59e78a7123a636e31ecafc9147046f190/telcoin-audit/contracts/swap/AmirX.sol#L86C13-L95C14]

External pre-conditions

No response

Attack Path

1. `defiToStablecoinSwap()` is called to swap from `USDC` to `XYZ1` (`ss.origin`), and from `XYZ1` to `XYZ2`.
2. the check for the `ss.oAmount` (`XYZ1` amount) is done, and the burn limit for that token hasn't been exceeded.
3. `_defiSwap()` is called to swap from `USDC` to `XYZ1`, where it returned `XYZ1` amount (`ss.oAmount`) greater than the initial one given in the inputs.
4. this new `ss.oAmount` hasn't been checked if it violates the burn limit, and the swap is done by burning an `ss.oAmount` amount of `XYZ1` from the user that has been received after the external swap done by the aggregator, and minting an `ss.tAmount` to the user.

Impact

The verification on the stablecoinSwap `ss.oAmount` parameter **is done before getting the actual amount of the `ss.oAmount` that is received when calling `_defiSwap()`**, so if the `_defiSwap()` call returns a balance difference of `XYZ1` (`ss.origin`) token different from the `ss.oAmount`; then the check made by the `_verifyStablecoinSwap()` to validate the amount of `XYZ1` token to be burnt might be violated if the actual modified `ss.oAmount` is greater than the `ss.oAmount` initially checked for.

PoC

No response

Mitigation

- Update `defiToStablecoinSwap()` to `_verifyStablecoinSwap` after the `_defiSwap()`:

```
    function defiToStablecoinSwap(
        address wallet,
        StablecoinSwap memory ss,
        DefiSwap memory defi
    ) external payable onlyRole(SWAPPER_ROLE) whenNotPaused {
        // checks if defi will fail
        _verifyDefiSwap(wallet, defi);
-       // checks if stablecoin swap will fail
-       _verifyStablecoinSwap(wallet, ss);


        //check balance to adjust second swap
        uint256 iBalance = ERC20(ss.origin).balanceOf(wallet);
        _defiSwap(wallet, defi);
        uint256 fBalance = ERC20(ss.origin).balanceOf(wallet);
        ss.oAmount = fBalance - iBalance;

+       // checks if stablecoin swap will fail
+       _verifyStablecoinSwap(wallet, ss);

        //change balance to reflect change
        _stablecoinSwap(wallet, ss);
    }
```

- Update `AmirX.swap()` for the same issue as well:

```
 function swap(
        address wallet,
        bool directional,
        StablecoinSwap memory ss,
        DefiSwap memory defi
    ) external payable onlyRole(SWAPPER_ROLE) whenNotPaused {
        // checks if it will fail
        if (ss.destination != address(0)) _verifyStablecoinSwap(wallet, ss);
        if (defi.walletData.length != 0) _verifyDefiSwap(wallet, defi);

        if (directional) {
            // if only defi swap
            if (ss.destination == address(0)) _defiSwap(wallet, defi);
            else {
                // if defi then stablecoin swap
                //check balance to adjust second swap
                uint256 iBalance = ERC20(ss.origin).balanceOf(wallet);
                if (defi.walletData.length != 0) _defiSwap(wallet, defi);
                uint256 fBalance = ERC20(ss.origin).balanceOf(wallet);
```

```
                //change balance to reflect change
                if (fBalance - iBalance != 0) ss.oAmount = fBalance - iBalance;
+               _verifyStablecoinSwap(wallet, ss);
                _stablecoinSwap(wallet, ss);
            }
        } else {
            // if stablecoin swap
            _stablecoinSwap(wallet, ss);
            // if only stablecoin swap
            if (defi.walletData.length != 0) _defiSwap(wallet, defi);
        }
    }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/telcoin/telcoin-audit/pull/60

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.