our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Möbius Exchange

SECURITY REVIEW

Date: 30 April 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, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

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 Möbius Exchange

Möbius is next-generation DeFi meets unparalleled capital efficiency. Unlike conventional AMMs, Möbius isn't just another swap protocol—it's a revolutionary liquidity engine designed to maximize returns, minimize slippage, and unlock seamless trading for stablecoins, LSTs, LRTs, and other pegged assets.

By utilizing a new approach to StableSwap by leveraging the Asset Liability Management (ALM) mode, Möbius provides liquidity pools that enable seamless swapping of pegged assets (e.g., stablecoins such as USDT, USDC, DAI, and 1:1 pegged assets such as WBTC and FBTC) and assets pegged to a gradually shifting ratio, such as LSTs/LRTs (e.g., ETH, mETH, cmETH). The design supports single-sided liquidity provisioning, ensuring minimal slippage and optimal swap rates.

With single-sided liquidity provision, extendable multi-asset pools, and precision-driven incentives, Möbius eliminates inefficiencies and supercharges liquidity across the Mantle ecosystem. By eliminating liquidity fragmentation and enabling extendable pools with more than two assets, Möbius maximizes capital efficiency, inefficient swaps, and enhances the user experience for both traders, and liquidity providers, and protocols that demand more.

Key Advantages of Möbius:

- `Single-Sided Liquidity Provision` – Simplifies liquidity provision by allowing LPs to deposit and withdraw a single asset, eliminating the need for exposure to multiple tokens.
- `Extendable Pool Design` – Reduces liquidity fragmentation by supporting multi-asset pools, enabling direct swaps without relying on inefficient multi-pool routing.
- `Demand-Driven TVL` – Token distribution in the pool naturally reflects market demand. For example, a pool could hold 5M USDT, 5M USDC, and 2M DAI, without artificially favouring swaps from DAI to other assets.
- `Fine-Tuned Emission Control` – Enables precise incentive targeting for specific assets, enhancing capital efficiency and improving overall liquidity depth.

- `Seamless DeFi Integrations` – Möbius's composable design allows DeFi aggregators and protocols to build on top of it, further boosting capital efficiency and expanding its ecosystem. Möbius LP tokens are yield-bearing assets that can integrate with other DeFi protocols, unlocking use cases such as collateralization, fixed-yield strategies and composability within lending markets and structured products.

Learn more about Mobius's design overview 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 was conducted over the span of 7 days by the core Shieldify team. The code is written professionally, and all best practices are considered. However, the security review identified one Medium-severity issue where a duplicate pool in the router path could bypass transaction constraints, leading to unintended behavior, and three Low-severity issues related to missing slippage protection, missing coverage ratio checks, and the owner's inability to update assets.

Shieldify extends its gratitude to the Möbius team for cooperating with all the questions our team had and strictly following the recommendations provided.

5.1 Protocol Summary

Project Name	**Möbius Exchange**
Repository	MobiusExchange-core
Type of Project	DEX, AMM
Audit Timeline	7 days
Review Commit Hash	21eeeca84ea26abfb131f758164256af8a706aaa
Fixes Review Commit Hash	5d5ff946371a62af1040f3e1bbd30a5fc620c8c4

5.2 Scope

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

File	**nSLOC**
src/asset/AggregateAccount.sol	15
src/asset/Asset.sol	125
src/asset/VariantAsset.sol	20
src/interfaces/IAsset.sol	3
src/interfaces/IMobiusRouter.sol	3
src/interfaces/IPool.sol	3
src/interfaces/IRelativePriceProvider.sol	3
src/pool/VariantPool.sol	387
src/pool/Core.sol	128
src/pool/StablePool.sol	363
src/router/MobiusRouter.sol	71
src/util/SingleCallPerTransactionBase.sol	19
Total	**1140**

6. Findings Summary

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

- **Medium** issues: **1**
- **Low** issues: **3**
- **Info** issues: **1**

ID	Title	Severity	Status
[M-01]	Duplicate Pool in Router Path Can Cause Transaction Failure	Medium	Fixed
[L-01]	Missing Slippage Protection in Deposit Function	Low	Acknowledged
[L-02]	Owner Cannot Update Asset With `addAsset()`	Low	Fixed
[L-03]	Missing Coverage Ratio Check in **quotePotentialWithdraw()** and **quotePotentialSwap()**	Low	Fixed
[I-01]	Integer Underflow in Haircut Calculation for High Decimal Tokens	Info	Fixed

7. Findings

[M-01] Duplicate Pool in Router Path Can Cause Transaction Failure

Severity

Medium Risk

Description

In the `_swap()` function in the `MobiusRouter` contract, there is an issue when the `poolPath` contains duplicate pool addresses. Each pool's `swap()` function is protected by the `conditionallySingleCallPerTransaction()` modifier, which is designed to prevent more than one call to the function per transaction:

```
function _swap(address[] calldata tokenPath, address[] calldata poolPath,
    uint256 fromAmount, address to)
    internal
    returns (uint256 amountOut, uint256 haircut)
{
// [code omitted for brevity]

    for (uint256 i; i < poolPath.length; ++i) {
// [code omitted for brevity]

// make the swap with the correct arguments
        (amountOut, localHaircut) = IPool(poolPath[i]).swap(
            tokenPath[i],
            tokenPath[i + 1],
            nextFromAmount,
            0, // minimum amount received is ensured on calling function
            nextTo,
            type(uint256).max // deadline is ensured on calling function
        );
// [code omitted for brevity]
    }
}
```

The conditionallySingleCallPerTransaction() modifier works by checking the gas consumed when accessing an address's balance:

```
modifier conditionallySingleCallPerTransaction() {
    if (coldAccessGasCost != -1) {
        address addressToCheck = address(uint160(bytes20(blockhash(block.
           number))));
        uint256 initialGas = gasleft();
        uint256 temp = addressToCheck.balance;
        uint256 gasConsumed = initialGas - gasleft();
        if (gasConsumed != uint256(coldAccessGasCost)) revert
           AlreadyCalledInThisTransaction();
    }
    _;
}
```

When a pool's swap() function is called multiple times in the same transaction, the second call will revert with the AlreadyCalledInThisTransaction error. This means that any router transaction with a poolPath containing duplicate pool addresses will fail. **But the protocol design allows pools to support multiple assets (as shown in StablePool and VariantPool), so this creates a limitation in constructing efficient swap paths.**

```
function addAsset(address token, address asset) external onlyOwner {
      if (token == address(0)) revert Pool_AddressShouldNotBeZero();
      if (asset == address(0)) revert Pool_AddressShouldNotBeZero();
      if (_containsAsset(token)) revert Pool_AssetAlreadyExists();

      _addAsset(token, VariantAsset(asset));

      emit AssetAdded(token, asset);
}
```

For example:

1. PoolA has USDC, USDT, and ETH, while PoolB only has USDT and ETH.
2. A user spots a price gap in ETH between the pools, creating an arbitrage opportunity. Their swap strategy is:
   - PoolA: USDC ETH
   - PoolB: ETH USDT
   - PoolA: USDT USDC
3. This is a valid arbitrage strategy, but MobiusRouter.swapTokensForTokens() can't execute it.

Location of Affected Code

File: src/router/MobiusRouter.sol

Impact

Any router transaction with a `poolPath` containing duplicate pool addresses will fail. But the protocol design allows pools to support multiple assets (as shown in `StablePool` and `VariantPool`), so this creates a limitation in constructing efficient swap paths.

Recommendation

If duplicate pools should not be supported, add a check in `MobiusRouter.swapTokensForTokens()` with a direct error. If they should be supported, remove `conditionallySingleCallPerTransaction()` from `pool.swap()`.

Team Response

Fixed. We've added validation to prevent duplicate pools in swap paths.

[L-01] Missing Slippage Protection in Deposit Function

Severity

Low Risk

Description

The pool contract has a vulnerability in its deposit mechanism. When a user deposits tokens into the pool, the contract does not provide a slippage protection mechanism. This is problematic because the exchange rate between deposited tokens and LP tokens can be manipulated through swaps.

In the `_deposit()` function, the LP token amount (liquidity) is calculated based on the current ratio of total supply to liability:

```
function _deposit(Asset asset, uint256 amountInWad, address to) private
   returns (uint256 liquidity) {
    uint256 totalSupply = asset.totalSupply();
    uint256 liability = asset.liability();

// Calculate amount of LP to mint : deposit * TotalAssetSupply /
   Liability
    if (liability == 0) {
        liquidity = amountInWad;
    } else {
        liquidity = _tokenAmountToLiquidity(amountInWad, liability,
           totalSupply);
    }

    if (liquidity == 0) revert Pool_DustAmount();

    asset.addCash(amountInWad);
    asset.addLiability(amountInWad);
    asset.mint(to, liquidity);
}
```

The issue stems from the `swap()` function, which increases an asset's liability through the `haircut` mechanism:

```
fromERC20.safeTransferFrom(address(msg.sender), address(fromAsset),
   fromAmount);
fromAsset.addCash(_toWad(fromAmount, fromAsset.underlyingTokenDecimals())
   );
toAsset.removeCash(actualToAmountInWad);
toAsset.addLiability(_dividend(haircutInWad, _retentionRatio));
toAsset.transferUnderlyingToken(to, actualToAmount);
```

When a swap occurs, the `toAsset`'s liability increases (through `addLiability()`) while its total supply remains unchanged. This causes the ratio of `totalSupply/liability` to decrease, which directly affects the amount of LP tokens received in subsequent deposits.

Location of Affected Code

File: src/pool/StablePool.sol

File: src/pool/VariantPool.sol

Impact

This vulnerability allows for the front-running of deposit transactions, resulting in users receiving fewer LP tokens than expected. It can lead to financial losses for depositors, as they have no way to specify a minimum acceptable amount of LP tokens when depositing.

Recommendation

Implement minimum liquidity checks for the deposit function, similar to what already exists for withdrawals and swaps.

Team Response

Acknowledged. We acknowledge that front-running could affect the amount of LP tokens received during a deposit. However, after careful consideration, we've decided not to implement slippage protection because the LP tokens always represent the correct underlying token value, ensuring no loss of principal even if front-run.

[L-02] Owner Cannot Update Asset With `addAsset()`

Severity

Low Risk

Description

In `StablePool`, there is a logical flaw in the `_addAsset()` function. The function contains a conditional block that is unreachable due to the way the function is called in the `addAsset()` method.

```
function _addAsset(address key, Asset val) private {
    if (_assets.inserted[key]) {
        _assets.values[key] = val;
    } else {
        _assets.inserted[key] = true;
        _assets.values[key] = val;
        _assets.indexOf[key] = _assets.keys.length;
        _assets.keys.push(key);
    }
}
```

The issue arises because every time `_addAsset()` is called from the `addAsset()` function, there is an explicit check using `_containsAsset(token)` that ensures the asset doesn't already exist in the pool:

```
function addAsset(address token, address asset) external onlyOwner {
    if (token == address(0)) revert Pool_AddressShouldNotBeZero();
    if (asset == address(0)) revert Pool_AddressShouldNotBeZero();
    if (_containsAsset(token)) revert Pool_AssetAlreadyExists();

    _addAsset(token, Asset(asset));

    emit AssetAdded(token, asset);
}
```

```
function _containsAsset(address key) private view returns (bool) {
    return _assets.inserted[key];
}
```

Since `_containsAsset()` checks `_assets.inserted[key]` and `addAsset()` reverts if this is true, the first condition in `_addAsset()` will never be reached. This branch might be for updating the asset, but the bug breaks that functionality.

Location of Affected Code

File: src/pool/StablePool.sol

Impact

The first condition in `_addAsset()` will never be reached; this branch might be for updating the asset, but the bug breaks that functionality.

Recommendation

Remove the first branch in `_addAsset()` or remove `_containsAsset(token)` in `addAsset()`.

Team Response

Fixed.

[L-03] Missing Coverage Ratio Check in `quotePotentialWithdraw()` and `quotePotentialSwap()`

Severity

Low Risk

Description

The `quotePotentialWithdraw()` and `quotePotentialSwap()` functions in the `VariantPool` and `StablePool` contracts are designed to provide estimates for withdrawal and swap operations without executing the actual transactions. However, unlike their actual execution counterparts (`withdraw()` and `swap()`), these quote functions do not verify if the post-operation coverage ratio meets the minimum threshold requirements.

In the actual execution functions:

- `withdraw()` verifies that after withdrawal, the coverage ratio is above `_rThreshold` (0.25e18 by default)
- `swap()` checks that after the swap, the coverage ratio of `toAsset` is above `_rThreshold`
- `withdrawFromOtherAsset()` verifies that after withdrawal, the coverage ratio is above `ETH_UNIT` (1e18)

Similar coverage ratio check is present in `quotePotentialWithdrawFromOtherAsset()`, but missing in `quotePotentialWithdraw()` and `quotePotentialSwap()`:

```
function quotePotentialWithdraw(address token, uint256 liquidity)
    external
    view
    whenNotPaused
    returns (uint256 amount, uint256 fee)
{
    if (liquidity == 0) revert Pool_InputAmountZero();
    VariantAsset asset = _assetOf(token);
    uint256 amountInWad;
    uint256 feeInWad;
    (amountInWad,, feeInWad) = _quoteWithdraw(asset, liquidity);
    amount = _fromWad(amountInWad, asset.underlyingTokenDecimals());
    fee = _fromWad(feeInWad, asset.underlyingTokenDecimals());
// Missing coverage ratio check
}
```

This inconsistency can lead to misleading quotes for users, who might receive quotes for operations that would actually revert when executed due to an insufficient coverage ratio.

Location of Affected Code

File: src/pool/VariantPool.sol

File: src/pool/StablePool.sol

Impact

This inconsistency can lead to misleading quotes for users, who might receive quotes for operations that would actually revert when executed due to an insufficient coverage ratio.

Recommendation

Add the missing coverage ratio checks to both quote functions

Team Response

Fixed.

[I-01] Integer Underflow in Haircut Calculation for High Decimal Tokens

Severity

Informational

Description

In `MobiusRouter`, there is an integer underflow vulnerability in the haircut calculation when tokens with more than 18 decimals are used. The issue occurs in the `_swap()` function where the haircut is calculated:

```
haircut += localHaircut * 10 ** (18 - IERC20Metadata(tokenPath[i + 1]).
    decimals());
```

When a token's decimal value exceeds 18, the expression `(18 - decimals)` becomes negative, causing an underflow in the exponentiation operation, which will revert the transaction.

Location of Affected Code

File: src/router/MobiusRouter.sol

Impact

The `MobiusRouter.swapTokensForTokens()` does not work for tokens with decimals larger than 18.

Proof of Concept

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract UnderFlow {

    function checkUnderFlow(uint256 decimal) public returns (uint256){
        uint256 haircut = 1e18;
        uint256 localHaircut = 1e18;
        haircut += localHaircut * 10 ** (18-decimal);
        return haircut;
    }
}
```

Deploy this contract in Remix, when `decimal==17`, it can run successfully, when `decimal == 19`, the tx will revert.

Recommendation

Modify the haircut calculation to handle tokens with more than 18 decimals:

```
if (IERC20Metadata(tokenPath[i + 1]).decimals() <= 18) {
    haircut += localHaircut * 10 ** (18 - IERC20Metadata(tokenPath[i +
        1]).decimals());
} else {
    haircut += localHaircut / 10 ** (IERC20Metadata(tokenPath[i + 1]).
        decimals() - 18);
}
```

Team Response

Fixed. The protocol will not support tokens with >18 decimals (as most tokens use 18 decimals). We've added a validation check during pool initialization to enforce this.

our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!