Summary

Audit Report prepared by Solidified covering the Mauve AMM.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on April 6, 2023, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repositories:

Repositories:

- https://github.com/violetprotocol/VioletID
- https://github.com/violetprotocol/ethereum-access-token
- https://github.com/violetprotocol/mauve-swap-router-contracts
- https://github.com/violetprotocol/mauve-core
- https://github.com/violetprotocol/mauve-periphery

Commits:

- `88b78870d657bd4d663aefa4e4d6075c45289512`
- `f0c8e6c96f5f96ef4716c4d42f2a2897887f478c`
- `05126b1f942d34bb053b3f97fd5e2650a98981fc`
- `b2243a02cae53db0d39e7acadab57902d24e1264`
- `9bcb305617e4c1c4728243cb22f567df605cd505`

The fixes were verified at the following commits:

- `0fdb4e57f5f2d9c111686613db0bf7ce94c77e10`
- `0357836cdde3350d57477ed69d8fde4ed78a63e7`
- `06b9ba8bbcc06e5e0509816cfeb7c7aa4b812d6b`
- `0ec6731f58e19da6d2f25d01d7a1d6c02642c2ba`
- `efcbb70d8059a0e48634a1d071e2bb6723ed2af4`

```
├── VioletID
│   └── contracts
│       ├── IVioletID.sol
│       └── VioletID.sol
├── ethereum-access-token
```

```
│   └── contracts
│       ├── AccessTokenConsumer.sol
│       ├── AccessTokenVerifier.sol
│       ├── IAccessTokenVerifier.sol
│       ├── KeyInfrastructure.sol
│       └── examples
│           └── DummyDapp.sol
├── mauve-swap-router-contracts
│   └── contracts
│       ├── MauveSwapRouter.sol
│       ├── SwapRouter.sol
│       ├── base
│       │   ├── ApproveAndCall.sol
│       │   ├── EATMulticallExtended.sol
│       │   ├── EATMulticallPeripheryPaymentsExtended.sol
│       │   ├── EATMulticallPeripheryPaymentsWithFeeExtended.sol
│       │   ├── ImmutableState.sol
│       │   ├── MauveCompliance.sol
│       │   ├── MulticallExtended.sol
│       │   ├── OracleSlippage.sol
│       │   ├── PeripheryPaymentsExtended.sol
│       │   ├── PeripheryPaymentsWithFeeExtended.sol
│       │   └── PeripheryValidationExtended.sol
│       ├── interfaces
│       │   ├── IApproveAndCall.sol
│       │   ├── IEATMulticallExtended.sol
│       │   ├── IImmutableState.sol
│       │   ├── IMauveFactoryReduced.sol
│       │   ├── IMauveSwapRouter.sol
│       │   ├── IMulticallExtended.sol
│       │   ├── IOracleSlippage.sol
│       │   ├── IPeripheryPaymentsExtended.sol
│       │   ├── IPeripheryPaymentsWithFeeExtended.sol
│       │   ├── IQuoter.sol
│       │   ├── IQuoterV2.sol
│       │   ├── ISwapRouter.sol
│       │   ├── ITokenValidator.sol
│       │   └── IWETH.sol
│       ├── lens
│       │   ├── Quoter.sol
│       │   └── QuoterV2.sol
│       └── libraries
│           ├── Constants.sol
│           └── PoolTicksCounter.sol
├── mauve-v3-core
│   └── contracts
│       ├── MauveFactory.sol
│       ├── MauvePool.sol
│       ├── MauvePoolDeployer.sol
│       ├── NoDelegateCall.sol
│       ├── interfaces
│       │   ├── IERC20Minimal.sol
│       │   ├── IMauveFactory.sol
│       │   ├── IMauvePool.sol
│       │   ├── IMauvePoolDeployer.sol
│       │   ├── IVioletID.sol
│       │   ├── callback
│       │   │   ├── IMauveMintCallback.sol
│       │   │   └── IMauveSwapCallback.sol
│       │   └── pool
│       │       ├── IMauvePoolActions.sol
│       │       ├── IMauvePoolDerivedState.sol
│       │       ├── IMauvePoolEvents.sol
│       │       ├── IMauvePoolImmutables.sol
│       │       ├── IMauvePoolOwnerActions.sol
│       │       └── IMauvePoolState.sol
│       └── libraries
│           ├── BitMath.sol
│           ├── FixedPoint128.sol
│           ├── FixedPoint96.sol
│           ├── FullMath.sol
│           ├── LiquidityMath.sol
│           ├── LowGasSafeMath.sol
│           ├── Oracle.sol
│           ├── Position.sol
│           ├── SafeCast.sol
│           ├── SqrtPriceMath.sol
│           ├── SwapMath.sol
│           ├── Tick.sol
│           ├── TickBitmap.sol
│           ├── TickMath.sol
│           ├── TransferHelper.sol
│           └── UnsafeMath.sol
└── mauve-v3-periphery
    └── contracts
        ├── NonfungiblePositionManager.sol
        ├── NonfungibleTokenPositionDescriptor.sol
        ├── SwapRouter.sol
        ├── base
        │   ├── BlockTimestamp.sol
        │   ├── EATMulticall.sol
        │   ├── EATMulticallPeripheryPayments.sol
        │   ├── EATMulticallPeripheryPaymentsWithFee.sol
        │   ├── ERC721Permit.sol
        │   ├── LiquidityManagement.sol
        │   ├── MauveCompliance.sol
        │   ├── Multicall.sol
        │   ├── PeripheryImmutableState.sol
        │   ├── PeripheryPayments.sol
        │   ├── PeripheryPaymentsWithFee.sol
        │   ├── PeripheryValidation.sol
        │   └── SelfPermit.sol
        ├── interfaces
        │   ├── IEATMulticall.sol
        │   ├── IERC20Metadata.sol
        │   ├── IERC721Permit.sol
        │   ├── IMulticall.sol
        │   ├── INonfungiblePositionManager.sol
        │   ├── INonfungibleTokenPositionDescriptor.sol
        │   ├── IPeripheryImmutableState.sol
        │   ├── IPeripheryPayments.sol
        │   ├── IPeripheryPaymentsWithFee.sol
        │   ├── IQuoter.sol
        │   ├── IQuoterV2.sol
        │   ├── ISelfPermit.sol
        │   ├── ISwapRouter.sol
        │   ├── ITickLens.sol
        │   └── external
        │       ├── IERC1271.sol
        │       ├── IERC20PermitAllowed.sol
        │       ├── IMauveFactoryReduced.sol
        │       ├── IVioletID.sol
        │       └── IWETH9.sol
        │
```

```
├── lens
│   ├── MauveInterfaceMulticall.sol
│   ├── Quoter.sol
│   ├── QuoterV2.sol
│   └── TickLens.sol
└── libraries
    ├── BytesLib.sol
    ├── CallbackValidation.sol
    ├── ChainId.sol
    ├── HexStrings.sol
    ├── LiquidityAmounts.sol
    ├── NFTDescriptor.sol
    ├── NFTSVG.sol
    ├── OracleLibrary.sol
    ├── Path.sol
    ├── PoolAddress.sol
    ├── PoolTicksCounter.sol
    ├── PositionKey.sol
    ├── PositionValue.sol
    ├── SqrtPriceMathPartial.sol
    ├── TokenRatioSortOrder.sol
    └── TransferHelper.sol
```

Intended Behavior

The code base implements a compliant non-custodial AMM that uses Violet’s compliance system.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	Medium-High	-
Test Coverage	High	-

Issues Found

Solidified found that the Mauve AMM contracts contain no critical issues, 3 major issues, 4 minor issues, and 6 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Ethereum Access Token: Reentrancy allows consuming the same token multiple times	Major	Resolved
2	Approvals of Non-Fungible Liquidity tokens are not restricted only to VioletID holders	Major	Resolved
3	SwapRouter: Token reentrancy can be abused to perform arbitrary calls	Major	Resolved
4	MauveFactory cannot be configured to work with both routers simultaneously	Minor	Acknowledged
5	Ethereum Access Token: Tokens can be used multiple times after hard forks	Minor	Resolved
6	Emergency mode cannot be disabled	Minor	Resolved
7	refundETH can be abused to perform arbitrary calls without authorization	Minor	Resolved
8	Violet ID: initialize is called in the constructor	Note	Resolved
9	Similar access control modifiers are implemented with different patterns	Note	Acknowledged
10	onlyActiveIssuer modifier is never used in the codebase	Note	Resolved
11	Wrong comment references to Uniswap instead of Mauve	Note	Resolved
12	Library imported from Uniswap instead of Mauve	Note	Resolved

	on token descriptor		
13	Gas optimizations	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. Ethereum Access Token: Reentrancy allows consuming the same token multiple times

The modifier `requiresAuth` first calls `verify`, which verifies the token and checks that it was not used before. Then, the code of the function is executed. Finally, `_consumeAccessToken` is called to mark the token as used. Because the token is only marked as used after the execution of the function, the same token can be used multiple times when the executed function allows reentrancy / performs an external call. When the function is called another time during its execution, the token will not be marked as used yet and the `requiresAuth` check, therefore, will succeed.

Recommendation

We recommend consuming the access token before the function execution, i.e. swapping lines 2 and 3 of the modifier.

2. Approvals of Non-Fungible Liquidity tokens are not restricted only to VioletID holders

In the Solidity contract `mauve-v3-periphery/contracts/NonfungiblePositionManager.sol` the functions `approve`, `setApprovalForAll` and the function `permit` inherited from `mauve-v3-periphery/base/ERC721Permit.sol` do not restrict approvals to only VioletID holders, contrary to what the comments on line `432` and `437` state. Allowing non-VioletID holders to be operators of a liquidity token.

Recommendation

We recommend adding `onlyAllowedToInteract(to)` on the three mentioned functions in order to implement this restriction.

3. `SwapRouter`: Token reentrancy can be abused to perform arbitrary calls

Similarly to the previous issue, reentrancy in the swap router can be abused to perform arbitrary unauthorized calls. The swap router performs calls to `pool.swap` with a recipient address that is provided by the user. Some tokens (e.g., ERC777) provide callback functionality for transfers. These can be abused by the user for reentrancy and to circumvent the access control.

Recommendation

We recommend using a more sophisticated protection mechanism. For example, instead of using a boolean flag to distinguish if the contract is in a multicall or not, the sequence could be encoded and consumed. Alternatively, encoding the number of allowed calls (and decrementing them each time) would also prevent this issue.

Minor Issues

4. `MauveFactory` cannot be configured to work with both routers simultaneously

The role system implemented in `mauve-v3-core/contracts/MauveFactory.sol` maps one `string`, representing a role, to an `address`. This makes it impossible for two addresses to hold the same role. The role `swapRouter` can only be set to one address simultaneously.

Therefore the Solidity contract `mauve-v3-core/contracts/MauvePool.sol` function `swap`, guarded by `onlySwapRouter` modifier, can only be accessible by one of the two routers of the protocol, either `mauve-v3-periphery/contracts/SwapRouter.sol` or `mauve-swap-router-contracts/SwapRouter.sol`, leaving the other one in a DOS state.

Recommendation

We recommend using OpenZeppelin role-based access control module instead of the current roles system because it supports multiple role holders which fits the protocol architecture needs since both routers need to have access to the pool.

5. Ethereum Access Token: Tokens can be used multiple times after hard forks

The Ethereum access token contract sets the `DOMAIN_SEPARATOR` (containing the chain ID) in the constructor and always uses this separator. This can be problematic when there is a hard fork that changes the chain ID (e.g., Ethereum -> Ethereum Classic) after the contract deployment. An access token will then be usable on both chains, although it is only intended for one of them. Furthermore, creating tokens for the new chain will not be possible as it still uses the old chain ID.

Recommendation

Consider using the OpenZeppelin ERC721 contract or caching the original chain ID and returning the cached `DOMAIN_SEPARATOR` if the chain id has not changed. This is OpenZeppelin's solution to this problem and a gas-efficient way to solve it, as the explicit construction is only required when the chain ID has changed (i.e., on hard forks).

6. Emergency mode cannot be disabled

In both instances of Mauve Compliance, `mauve-v3-periphery/contracts/base/MauveCompliance.sol` and `mauve-swap-router-contracts/contracts/base/MauveCompliance.sol` emergency mode can be activated using the function `activateEmergencyMode` but it cannot be deactivated leaving both the swap router and the position manager stuck in emergency mode forever after this one is activated.

Recommendation

We recommend changing the function `activateEmergencyMode()` of `MauveCompliance` to a more general function `setEmergencyMode(bool mode)` that gives the owner the ability to both enable and disable emergency mode.

7. `refundETH` can be abused to perform arbitrary calls without authorization

The function `refundETH` in `ETAMulticallPeripheryPaymentsWithFee` performs a call to the callee in order to transfer the ETH. When it is called from within a multicall, `isMulticalling` will be set to true. Therefore, an attacker can use this callback to call any other function with an `onlySelfMulticall` modifier. Because of this, granting a token to a multicall that contains a `refundETH` is equivalent to granting the user access to all functions and granular access control is no longer possible.

Currently, this is not exploitable because only EOAs can interact with the AMM. However, because support for smart contracts may be added in the future, it is still advised to fix the problem.

Recommendation

Like in the previous issue related to reentrancy, we recommend using a more sophisticated protection mechanism.

Informational Notes

8. Violet ID: `initialize` is called in the constructor

In the constructor of `VioletID`, the `initialize` function is called. This is not recommended for contracts that are intended to be deployed behind a proxy. While there is no immediate security risk, it means that the implementation contract (which should only contain the code) will also have initialized storage and appear in block explorers as an NFT. This may confuse users that interact with the contract.

Recommendation

Consider only disabling the initializers for the implementation contracts, but not calling the initializer.

9. Similar access control modifiers are implemented with different patterns

In the contract `mauve-v3-core/contracts/MauveFactory.sol`, the access control modifiers `onlyPoolAdmin` and `onlyOwner` are implemented with two different patterns: `onlyPoolAdmin` performs the access control check directly in the modifier, whereas `onlyOwner` calls an internal function to perform the check. No different functionality is provided by any of the two patterns. Similarly, in the contract `mauve-v3-periphery/contracts/PeripheryValidation.sol` the modifier `checkDeadline` is implemented as a modifier that calls an internal function, which is an unnecessary change from the Uniswap v3 codebase since it adds no functionality.

Recommendation

In order to prevent future problems for developers we recommend implementing both factory modifiers with the same pattern.

Regarding the periphery validation `checkDeadline` modifier we recommend refactoring it only as one modifier without having to call an internal `_checkDeadline` function, in order to reduce the gas spent by the user.

10. `onlyActiveIssuer` modifier is never used in the codebase

In contract `ethereum-access-token/contracts/KeyInfrastructure.sol` the modifier `onlyActiveIssuer` is never used in the contract. This introduces dead code in the codebase.

Recommendation

We recommend removing this modifier from the contract in order to keep the code clean.

11. Wrong comment references to Uniswap instead of Mauve

There are some wrong comment references to Uniswap instead of Mauve in the following contracts:

- `mauve-v3-core/contracts/interfaces/IMauveFactory.sol:5`
- `mauve-v3-core/contracts/MauveFactory.sol:12`
- `mauve-v3-periphery/deploys.md:3`

12. Library imported from Uniswap instead of Mauve on token descriptor

In the contract `mauve-v3-periphery/contracts/NonfungibleTokenPositionDescriptor.sol` the library `SafeERC20Namer.sol` is imported from the uniswap npm package instead of the `mauve-v3-periphery/contracts/libraries` folder like the rest of libraries. This introduces unnecessary dependencies in the codebase.

Recommendation

We recommend to include the library `SafeERC20Namer.sol` inside the `libraries` folder to eliminate dependencies on the Uniswap v3 codebase.

13. Gas optimizations

There are a few optimizations that could be implemented to optimize the gas usage of the contracts:

1. `AccessTokenConsumer._verifier`, `AccessTokenVerifier.DOMAIN_SEPARATOR`, `KeyInfrastructure._root`: Because these variables are set in the constructor and not changeable, they could be marked as `immutable`. This would be especially beneficial for the access token, as it would save an SLOAD whenever the `requiresAuth` modifier is used, resulting in savings throughout the Mauve codebase and for future projects that use `AccessTokenConsumer`.
2. The modifier `multicalling` within `EATMulticall` always sets the variable `isMulticalling` from 0 (`false`) to 1 (`true`) and vice-versa. 0 -> x SSTOREs are much more expensive than x -> y (with x > 0, y > 0) transitions. Although there is a refund for x -> 0 transitions, this refund is capped. Furthermore, using a `uint256` would be cheaper because no masking is needed. Consider therefore using a `uint256` and the values 1 and 2. OpenZeppelin's `ReentrancyGuard` does this for the reasons mentioned above.
3. calldata instead of memory can be used for function arguments that do not get mutated on `AccessTokenVerifier` lines `42`, `55`, `68` and `73`. This makes it so that the data is not automatically loaded into memory.
4. Keeping revert strings under 32-bytes prevents the string from being stored in more than one memory slot. There are three instances of this issue: `AccessTokenVerifier:21`, `KeyInfrastructure:20` and `KeyInfrastructure:25`.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of DeFi Labs GmbH or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH