Solidly V3 AMM Security Review

Pashov Audit Group

Conducted by: pashov

September 22nd, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Solidly V3 AMM** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Solidly V3 AMM

The Solidly V3 AMM is a Uniswap V3 fork. It's goal is to be more gas efficient than Uniswap's implementation, as well as use a different fees mechanism. Some key differences between Solidly V3 and Uniswap V3 are:

- swap fees for a pool can be adjusted dynamically with the permissioned `feeSetter` role
- LPs don't accrue fees directly on-chain, all fees go to the permissioned `feeCollector` role and then they are processed off-chain to create a Merkle tree for on-chain claims
- the oracle logic is removed altogether
- no callbacks on swaps, LP mints & LP burns, only on `flash`

Observations

Uniswap V3 had two types of fees:

1. A swap fee that goes to LPs
2. A protocol fee, which is a percentage of the swap fee, sent to the Uniswap Treasury (set to 0 by default)

In Solidly V3 AMM there is only one type of fee - a swap fee that gets collected by the `feeCollector` role. According to the team this address will be the `RewardsDistributor` contract, expected source can be seen here. The permissioned role on the RewardDistributor periodically updates merkle roots to reflect the latest rewards earned by LPs and voters.

As the `SolidlyV3Factory` owner and the `RewardsDistributor` owner have control over fees distribution, even though they are implemented (as per the team) behind a timelock + multisig, a malicious or compromised multisig could take the swap fees

You can't deploy multiple A & B token pools with the same tick spacing values, but you can deploy multiple such pools with same swap fee values.

Privileged Roles & Actors

- Factory owner - can enable creation of pools with new `fee` + `tickSpacing` configuration (same as Uniswap) and also give/revoke the fee setter & collector roles
- Fee setter - can update the fee of any pool up to 10%
- Fee collector - can collect all the swap fees accrued for any pool

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **- 819283d2e400b697413b35771bd4957863d1fd71**

fixes review commit hash **- f8201ab299f976a8e0b0e0539e5a907cc83236bd**

Scope

The following smart contracts were in scope of the audit:

- `SolidlyV3Factory`
- `SolidlyV3Pool`
- `SolidlyV3PoolDeployer`
- `libraries/Position`
- `libraries/Status`
- `libraries/Tick`

7. Executive Summary

Over the course of the security review, pashov engaged with Solidly V3 AMM to review Solidly V3 AMM. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Solidly V3 AMM
Date	September 22nd, 2023

Findings Count

Severity	**Amount**
Medium	2
Low	1
Total Findings	**3**

Summary of Findings

ID	**Title**	**Severity**	**Status**
[M-01]	Pools are incompatible with non-standard ERC20 tokens	Medium	Resolved
[M-02]	Protocol owner has control over swap fees earned	Medium	Resolved
[L-01]	Fee value is not validated in constructor	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Pools are incompatible with non-standard ERC20 tokens

Severity

Impact: Medium, as it limits the possible pools in the app and with some special ones can result in loss of funds

Likelihood: Medium, as some of the tokens are not uncommon, like `USDT`

Description

The Solidly V3 AMM code removes the balance checks before and after transfers and also the `TransferHelper` library, calling the `transfer` and `transferFrom` methods of ERC20 directly. This will be problematic with three different types of tokens:

1. The `USDT` and `BNB` tokens do not return a `bool` on `transfer`, meaning transactions that add liquidity for a pool of those tokens would directly revert, resulting in limitation of the possible pools in the application
2. The `ZRX` and `EURS` tokens do not revert on failure, but instead return `false` - the return value of the methods is not checked, so this can result in a loss of funds if either `transfer` or `transferFrom` fails
3. Some tokens have a transfer fee (like `STA`, `PAXG`) and for example `USDT` has a fee mechanism but it's currently switched off - such tokens will not work correctly with the pools and can lead to a loss of funds

Recommendations

While the docs should state that fee-on-transfer tokens are not supported, the application is incompatible with other tokens as well - either state this in the

docs or consider using the `TransferHelper` library, also add the before and after transfer balance checks back.

Discussion

pashov: Tokens that do not return a `bool` on `transfer` or do not revert on failure are now correctly handled. Fee on transfer tokens are explicitly not supported.

[M-02] Protocol owner has control over swap fees earned

Severity

Impact: High, as it can result in a loss of yield for LPs

Likelihood: Low, as it requires a compromised or malicious owner

Description

In the Solidly V3 AMM the swap fees can be collected by an account that is given the `feeCollector` role. The `SolidlyV3Factory` can give and revoke this role to/from accounts. It is expected that the initial role holder will be set to a `RewardsDistributor` contract (expected source code can be seen here). In that contract, a permissioned `owner` will be able to set the Merkle root that shows who is able to claim fees (and how much).

There are a multiple possible options for the owner to misappropriate the swap fees from LPs:

1. The `SolidlyV3Factory` owner can set his own address or another one he controls to hold the `feeCollector` role, then just claim all fees
2. The `RewardsDistributor` owner can set a Merkle root for claimable rewards that holds only his address or another one he controls, then just claim all fees
3. The `RewardsDistributor` owner might never set a Merkle root, leaving the fees stuck in the application forever

Recommendations

While the team has communicated that all permissioned addresses will be behind a multi-sig and a Timelock contract, this is still an attack vector that can be exploited either from a compromised or a malicious owner. The Timelock has a 24h delay, which might be too low for users to claim their unclaimed yield. It's also possible that the Merkle root is set so they can't claim at all - you can think about a different mechanism for claiming rewards here.

Discussion

Solidly Team As per protocol design specification. For cases of malicious governance, our time-lock contract (OpenZeppelin) would warn users ahead of time. Acknowledged.

8.2. Low Findings

[L-01] Fee value is not validated in constructor

The pool fee value is checked in the `setFee` method and in `enableFeeAmount` as well with `require(fee <= 100000);`, but this check is omitted in the `SolidlyV3Pool` constructor, which means the initial fee can be set to a larger value. The check should be present in the constructor of `SolidlyV3Pool` as well.

Discussion

Solidly Team Acknowledged.