Mooniswap-v2 Audit

December 2020

By CoinFabrik

Introduction 3

Summary 3

Contracts 3

Analyses 4

Detailed findings 5

Severity Classification 5

Issues Found by Severity 6

Critical severity 6

Medium severity 6

Minor severity 6

Solution: 6

Enhancements 7

Solidity version is not fixed 7

Mixing ERC20 and Mooniswaps in the same array 7

Error in require message string at freezeEpoch 7

IRewardDistributionRecipient.sol should be called RewardDistributionRecipient.sol 8

Missing testing coverage for some contracts 8

Observations 8

Possible funds withdrawal by owner 8

Assumption zero address as the Ether address 8

Other specific analyses performed 9

Conclusion 10

Introduction

CoinFabrik was asked to audit the contracts for the Mooniswap project. First, we will provide a summary of our discoveries and then we will show the details of our findings.

Summary

The contracts audited are from the Mooniswap-v2 repository. The audit is based on the commit 355ee41415aa4ab97cccb18b77004caa03c07ac3 and updated to reflect changes at c910a4d82c18bd8a6670c0732f8714effb19578f.

Contracts

The audited contracts are:

- Mooniswap.sol
- MooniswapConstants.sol
- MooniswapDeployer.sol
- MooniswapFactory.sol
- ReferralFeeReceiver.sol
- governance/BaseGovernanceModule.sol
- governance/GovernanceFeeReceiver.sol
- governance/MooniswapFactoryGovernance.sol
- governance/MooniswapGovernance.sol
- governance/rewards/IRewardDistributionRecipient.sol
- governance/rewards/Rewards.sol
- inch/GovernanceMothership.sol
- interfaces/IGovernanceModule.sol
- interfaces/IMooniswapDeployer.sol
- interfaces/IMooniswapFactoryGovernance.sol
- interfaces/IReferralFeeReceiver.sol
- libraries/LiquidVoting.sol
- libraries/Sqrt.sol

- libraries/UniERC20.sol
- libraries/VirtualBalance.sol
- libraries/Vote.sol
- utils/BalanceAccounting.sol
- utils/Converter.sol

Mooniswap is an automatic market maker that implements an enhancement so the short term impact of large slippages is more favourable for liquidity providers and less for arbitrageurs and miners. See the Mooniswap whitepaper for the mathematical model details.

In the second version, a set of governance contracts were added to allow participants to vote parameters such as fee and slippage delay.

Analyses

The following analyses were performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Detailed findings

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

No issues of critical severity have been found.

Medium severity

No issues of medium severity have been found.

Minor severity

Incorrect user epoch assignation

The user can claim his accumulated rewards by epoch calling the *claim* function that then calls the *_claimOldEpoch* which calculates the *unclaimedtokens* and sets *lastUnprocessedEpoch* to the *currentEpoch* even if the user has not claimed all his pending epochs.

```
if (lastUserEpoch < _tokenInfo.lastUnprocessedEpoch) {
            unclaimedTokens =
                _tokenInfo.epochBalance[lastUserEpoch].inchBalance
                .mul(_userInfo.share[mooniswap][lastUserEpoch])
                .div(_tokenInfo.totalSupply[lastUserEpoch]);

            _userInfo.lastUnprocessedEpoch[mooniswap] =
currentEpoch;
```

Solution:

Instead setting the `lastUnprocessedEpoch` to the *currentEpoch* , which is the last epoch of the pool, it can be set to:

```
_userInfo.lastUnprocessedEpoch[mooniswap] =
       _userInfo.lastUnprocessedEpoch[mooniswap].add(1);
```

The code was refactored at commit 9e9cfcc8bf91f2c000d57b00a17df7c40eda2b47.

Enhancements

Solidity version is not fixed

The solidity files use semantic versioning to indicate they support any compiler version from the 0.6 releases. We recommend explicitly setting a fixed version.

```
pragma solidity 0.6.12;
```

To minimize possible changes introduced in a future solidity version that the team will not be aware of at deploying time. Also it is easier for a third party to validate the bytecode once it was deployed in the blockchain.

Mixing ERC20 and Mooniswaps in the same array

In the functions *_swap* and *_maxAmountForSwap* at Converter.sol the parameter *path* contains both ERC20 and Mooniswap addresses.

It is correct syntactically since Mooniswap inherits from ERC20. But it is not a good practice to mix objects that will be used differently.

Items at odd positions are used as Mooniswap

```
Mooniswap mooni = Mooniswap(address(path[i]));
```

And items at even positions are used as ERC20

```
amount = mooni.swap{value: value}(path[i-1], path[i+1], amount, 0,
address(this));
```

The code was updated to only require IERC20 addresses in the path parameter at commit 3c34a7658964d5b97675346c3a051e7ba2282141.

Error in require message string at freezeEpoch

In the function *freezeEpoch* at ReferralFeeReceiver.sol, the message at the require statement has a typing error. It should be “finalized” instead of “finlazed”.

```
require(info.lastUnprocessedEpoch == currentEpoch, "Previous epoch
is not finalized");
```

> ***The message string was fixed at commit 9e9cfcc8bf91f2c000d57b00a17df7c40eda2b47.***

IRewardDistributionRecipient.sol should be called RewardDistributionRecipient.sol

To comply with the standards of the Solidity file names the `I` should be removed from a contract because it is not an interface.

```
// -- CoinFabrik: Don't use I prefix for abstract contracts --
abstract contract RewardDistributionRecipient is Ownable {
     address public rewardDistribution;
```

> ***The code was refactored at commit to extract rewards into a separate base contract 3c778388ce853a4d746cd63ec95f992ec19738bf.***

Missing testing coverage for some contracts

The coverage is good with most contracts above 85%. Notably, there are a few contracts that do not have any coverage such as ReferralFeeReceiver.sol and GovernanceMothership.sol.

We suggest adding coverage for those contracts.

> ***Unit tests and coverage for ReferralFeeReceiver and GovernanceMothership were added at commits 854c4777a2b9b7d5ea4697b664e160e8f2591beb and c1401db0d82b2ab33d5641a2d1eea31ec0a88c3a.***

Observations

Possible funds withdrawal by owner

There is a function called *rescueFunds* that allows the owner to withdraw all the pool's token at any time, this address is set in the factory constructor and cannot be changed.

Users should be aware and trust this address since it has the potential to take all users funds if the private key gets compromised.

Recommendation: use a timelock mechanism to delay owner`s actions, so users can know in anticipation and take their tokens before this takes place.

> ***The function rescueFunds has protection to avoid the owner to remove tokens from the pool's tokens. It can only rescue tokens that were mistakenly sent.***

Assumption zero address as the Ether address

In the UniERC20 library the zero address (`0x0000000000000000000000000000000000000000`) is used to indicate operations with Ether. It is common for exchanges to use this address to facilitate uniform access to their functionality but still allowing Ether operations. One disadvantage of this particular value is that the EVM returns the same value when accessing uninitialized storage. For this reason other contracts like AAVE use `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` instead.

Using the zero address has the advantage that transactions will pay a little less since input data will have more zeros. We have not found possible attacks using the zero address so it should be safe to use. For inter compatibility with other contracts you may also consider `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` as a Ether address.

Other specific analyses performed

- Possible out of bound access in function *_maxAmountForSwap*. The function *_maxAmountForSwap* at Converter.sol expects the parameter path to have an odd number of elements, but it never checks that requirement. In solidity

0.6, this error may cause an out of bound access and consumption of all remaining gas in the transaction. The function explicitly skips the last step so `mooniswap.getReturn(path[i-1], path[i+1], stepAmount)` will not cause an out of bound access. So an error in the array will not cause excessive gas consumption.

- Default value on function *_updateTransfer* at MooniswapGovernance.sol. The value assigned to defaultValue in most cases is the return value of defaultValueGetter. There is a special case when both sender and receiver have voted and sender has not transferred all of their funds, in that case the 0 is assigned to defaultValue. In that case *updateBalance* in LiquidVoting.sol never uses the default value. The default value is only used as a vote when they have not voted. We understand the zero assignment is done to save the cost of calling the defaultValueGetter whose return value will not be used in that case.
- We examined the cases in which we pass as a parameter an array with unexpected contents such as an empty one, one with duplicated values and one with overflowed length to check that the security of the function is not compromised. Some of the functions in which we experimented with this sort of arrays are: notifyStakesChanged, batchNotifyFor and deposit.

Conclusion

The code is very well designed and written despite the fact that there are many innovations and new mechanisms. It doesn’t have much documentation in NatSpec format, but it has a few comments on key places.

The unit test worked and the coverage is adequate, but there are a few files that were not tested.

Overall the contracts are very good. We have not found serious issues, only minor ones. There are missing tests but they should be easy to complete.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Mooniswap-v2 project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.