SOLIDIFIED

Audit Report for Charm Finance - October 27, 2020

Summary

Audit Report prepared by Solidified covering the Charm Finance smart contracts (and their associated components).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on October 22th, 2020, and the results are presented here.

Update: The team submitted an update on October 25th. These fixes are reflected in this report.

Audited Files

The following contracts were covered during the audit:

```
contracts
├── CharmToken.sol
├── OptionsMarketMaker.sol
├── OptionsToken.sol
├── UniswapOracle.sol
├── libraries
│   ├── ABDKMath64x64.sol
│   └── UniERC20.sol
└── rewards
    ├── SeedRewards.sol
    └── StakingRewards.sol
```

Supplied in the following public source code repository:

https://github.com/Charm-Finance/charm-options

Notes

The audit was based on commit `7eb502b80a3f9b54aa059235e5655cbeea79f504`

Fixes were supplied in commit `2d8d670fcbbcad79b1d962d1e1856c9af9615f19`

Intended Behavior

The smart contract implements a market maker for options on Ethereum. Options are represented as ERC-20 tokens. Uniswap is used as a pricing Oracle.
The codebase also includes a staking and reward mechanism for a future governance token.

Executive Summary

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	High	-
Test Coverage[1]	Medium-High	-

[1] Automated coverage report attached as an appendix.

Issues Found

Solidified found that the Charm Finance contracts contain no critical issue, 1 major issues, 4 minor issue, in addition to 6c informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as it refers to best practices.

Issue #	Description	Severity	Status
1	OptionsMarketMaker has no oracle verification	Major	Acknowledged
2	Ownership privileges cannot be revoked in staking contract	Minor	Acknowledged
3	Consider additional sanity checks and automated oracle deployment for extra safety	Minor	Acknowledged
4	SeedRewards.sol: function exit() should allow users to specify minAmountOut	Minor	Resolved
5	UniERC20.sol: uniTransfer() can potentially fail if it transfers ETH to a smart contract	Minor	Resolved
6	Beware of malicious tokens	Note	Optional
7	Check for incorrect token in ERC20 recover function	Note	Resolved
8	Consider checking for zero amounts before minting	Note	Resolved
9	OptionsMarketMaker.sol: Consider calling settle() from redeem() if it has not been called yet	Note	Optional
10	OptionsMarketMaker.sol: Consider Providing contract users with an amountInRequired() convenience function	Note	Optional
11	Notes regarding OptionsMarketMaker owner privileges	Note	Optional

Critical Issues

No critical issues have been found.

Major Issues

1. `OptionsMarketMaker` has no oracle verification

Both the `OptionsMarketMaker` contract constructor and the `setOracle()` function do not verify that the given oracle is valid for the token pair represented by the options market maker.

This could lead to loss of funds in case an incorrect/malicious oracle is set, and it could also prevent settlement altogether in case the given oracle represents an invalid or a non-existent Uniswap pair (in which case `oracle.read()` will always return zero, and hence `OptionsMarketMaker.settle()` will always revert).

Recommendation
Add a new `pair` public member variable to the `OptionsMarketMaker` contract, and require assigning it via the constructor. Whenever a new oracle is assigned, require that `OptionsMarketMaker.pair == oracle.pair`.

Also, consider having `OptionsMarketMaker` deploy its own `UniswapOracle` based on the `pair` it receives in the constructor.

Team Response
“We wanted to keep the code flexible so that it works with any type of oracle in the future. For example, there might be Chainlink oracles which wouldn't have a Uniswap pair. We prefer the approach of giving privileges to the owner for the first version, to ensure it works as intended and to protect user deposits”

Minor Issues

2. Ownership privileges cannot be revoked in staking contract

The options market maker and staking mechanisms are implemented as contracts with special ownership privileges for emergency situations. The documentation states that these privileges exist for security purposes and will be removed in the future.
However, the implementation of this in `StakingReward.sol` does not allow ownership to be renounced. On ownership transfer, a new owner must acknowledge acceptance, so simply transferring ownership to `address(0)` is not possible.

Recommendation
Implement functionality to renounce ownership or use the already imported OpenZeppelin version for this.

Team Response
“We wish to keep StakingRewards as similar as possible to the original Synthetix code as any change increases the chance of a bug and makes it harder for people reading the code to check it. The reason we didn't feel the need to add renounceOwnership() was the contract has a finite duration and the reward token has a finite supply. After the rewards have been distributed, the deployed contracts will no longer be used.”

3. Consider additional sanity checks and automated oracle deployment for extra safety

Several parameters can be configured in a way that does not make sense or could lead to problems. This includes, for example, a start time set in the very far future or the decimals are incorrect for the token. Another example is a check for the Oracle to be funded so that users are incentivized to call the `snapshot()` function.

Recommendation
Consider including additional checks and deploying the oracle contract from the option market maker.

Team Response

“Fixed the possibility of incorrect decimals in UniswapOracle by retrieving them from the token instead of passing them in. Other sanity checks will be implemented in deployment scripts”

4. SeedRewards.sol: function exit() should allow users to specify minAmountOut

Function `exit()` currently does not allow users to specify the `minAmountOut` they’re willing to accept while exiting, and instead sets the minimum at zero. This could result in users getting unfavorable trades at the time of exit in case slippage is too high.

Recommendation
Have users specify `minAmountOut` as a parameter to `exit()`, similar to what is already the case with `withdraw()`.

Update: Fixed

5. UniERC20.sol: uniTransfer() can potentially fail if it transfers ETH to a smart contract

Function `uniTransfer()` uses `transfer()` to send back ETH to the `(to)` parameter, which only forwards 2300 gas. In cases where `(to)` is a smart contract who’s fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of preventing smart contracts (e.g. DAOs) from receiving any transfers, both in the `OptionsMarketMaker` and the `SeedRewards` contracts.

For a more in depth discussion of issues with `transfer()` and smart contracts, please refer to: `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation
Replace instances of `transfer()` with `call()` (all functions currently calling `uniTransfer()` are `nonReentrant`, so reentrancy attacks are not a concern).

Update: Fixed

Notes

6. Beware of malicious tokens

It is possible to design contracts that conform to the ERC20 specification but do not implement it accordingly to the desired behaviour.

With systems that allow for any token to be used, there's always the possibility of fake ones that either use the name and symbol of another famous one, as well some that contain malicious code. This impersonation attack is quite common on open DEX's.

Recommendation
There are two options for solving this, either create a list of allowed base tokens or leave it open and implement warnings in the interface to let users know they are dealing with a non-approved token.

7. Check for incorrect token in ERC20 recover function

`StakingReward.sol` and `SeedRewards.sol`: The function `recoverERC20()` checks the provided token address for the SNX token symbol, which is likely due to code reuse. This check is not applicable to this project.

Recommendation
Substitute the token symbol or remove the check.

Update: Fixed

8. Consider checking for zero amounts before minting

The `buy()` and `sell()` functions of `OptionMarketMaker.sol` always calls mint on both long and short order. Very often, this may not be the case, causing unnecessary gas cost.

Recommendation
Consider including checks for `> 0` before calling mint.

Update: Fixed

9. OptionsMarketMaker.sol: Consider calling settle() from redeem() if it has not been called yet

As a convenience to users, when a user calls `redeem()`, consider calling the `settle()` function if it has not been called yet.

10. OptionsMarketMaker.sol: Consider Providing contract users with an amountInRequired() convenience function

Consider providing a convenience function that would allow users of `OptionsMarketMaker` to determine how much of `baseToken` is required for a successful `buy()` call. In addition, also provide a function that would allow users to know the minimum amount received when calling `sell()`.

Similarly, consider adding the same convenience functions to the `SeedRewards` contract, where users are able to determine the amount to send to `stake()`, and the min amount to expect from `withdraw()`.

11. Notes regarding OptionsMarketMaker owner privileges

Although the Charm Finance team has noted that these owner privileges will be removed in future versions, and that they are only to be used in emergency situations, it is still worth noting what these privileges are in this report.

The current `OptionsMarketMaker` owner privileges allows them to perform the following operations:

- Pause the contract indefinitely. This includes preventing users from buying, selling, and redeeming their options.

- Change the contract’s oracle. If an invalid or malicious oracle is provided, users can potentially lose funds.
- Change the contract’s expiry date.
- Prematurely settle the contract before its expiry date.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Charm Finance 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.

Solidified Technologies Inc.

Appendix - Test Coverage Report

```
contract: CharmToken - 63.7%
  CharmToken.addMinter - 100.0%
  CharmToken.mint - 100.0%
  CharmToken.removeMinter - 100.0%
  CharmToken.setGovernance - 100.0%
  Context._msgSender - 100.0%
  ERC20.balanceOf - 100.0%
  ERC20.name - 100.0%
  ERC20.symbol - 100.0%
  ERC20.transfer - 100.0%
  ERC20._mint - 75.0%
  ERC20._transfer - 75.0%
  SafeMath.add - 75.0%
  SafeMath.sub - 75.0%
  ERC20.approve - 50.0%
  ERC20._approve - 0.0%
  ERC20.allowance - 0.0%
  ERC20.decimals - 0.0%
  ERC20.decreaseAllowance - 0.0%
  ERC20.increaseAllowance - 0.0%
  ERC20.totalSupply - 0.0%
  ERC20.transferFrom - 0.0%

contract: MockOracle - 100.0%
  MockOracle.getPrice - 100.0%
  MockOracle.setPrice - 100.0%

contract: MockToken - 68.3%
  Context._msgSender - 100.0%
  ERC20.approve - 100.0%
  ERC20.balanceOf - 100.0%
  ERC20.transfer - 100.0%
  ERC20.transferFrom - 100.0%
  MockToken.mint - 100.0%
  SafeMath.sub - 100.0%
  ERC20._approve - 75.0%
  ERC20._mint - 75.0%
  ERC20._transfer - 75.0%
  SafeMath.add - 75.0%
  ERC20.allowance - 0.0%
  ERC20.decimals - 0.0%
  ERC20.decreaseAllowance - 0.0%
  ERC20.increaseAllowance - 0.0%
  ERC20.name - 0.0%
  ERC20.symbol - 0.0%
  ERC20.totalSupply - 0.0%
```

```
contract: MockUniswapV2Pair - 100.0%
  MockUniswapV2Pair.getReserves - 100.0%
  MockUniswapV2Pair.setBlockTimestampLast - 100.0%
  MockUniswapV2Pair.setPrice0CumulativeLast - 100.0%
  MockUniswapV2Pair.setPrice1CumulativeLast - 100.0%
  MockUniswapV2Pair.setReserve0 - 100.0%
  MockUniswapV2Pair.setReserve1 - 100.0%

contract: OptionsToken - 63.5%
  Context._msgSender - 100.0%
  ERC20.balanceOf - 100.0%
  ERC20.name - 100.0%
  ERC20.symbol - 100.0%
  ERC20.totalSupply - 100.0%
  ERC20.transfer - 100.0%
  OptionsToken.burn - 100.0%
  OptionsToken.mint - 100.0%
  SafeMath.sub - 100.0%
  ERC20._burn - 75.0%
  ERC20._mint - 75.0%
  ERC20._transfer - 75.0%
  SafeMath.add - 75.0%
  ERC20.approve - 50.0%
  ERC20._approve - 0.0%
  ERC20.allowance - 0.0%
  ERC20.decimals - 0.0%
  ERC20.decreaseAllowance - 0.0%
  ERC20.increaseAllowance - 0.0%
  ERC20.transferFrom - 0.0%

contract: UniswapOracle - 74.5%
  Address.functionCall - 100.0%
  Address.isContract - 100.0%
  SafeERC20.safeTransfer - 100.0%
  UniswapOracle.claimReward - 100.0%
  UniswapOracle.takeSnapshot - 100.0%
  UniswapOracle.fetchSpotAndCumulativePrice - 90.0%
  SafeMath.mul - 87.5%
  SafeERC20._callOptionalReturn - 75.0%
  SafeMath.add - 75.0%
  SafeMath.div - 75.0%
  SafeMath.sub - 75.0%
  Address._functionCallWithValue - 50.0%
  UniswapOracle.getPrice - 50.0%
  Math.min - 0.0%
```