SOLIDIFIED

Audit Report for Unitrade on October 6, 2020

Summary

Audit Report prepared by Solidified covering the Unitrade smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on October 2nd, 2020.
The team provided an updated version on October 5th, 2020 and changes are reflected in this report.

Audited Files

The following contracts were covered during the audit:

- IUniTradeStaker.sol
- UniTradeIncinerator.sol
- UniTradeOrderBook.sol

The UniTrade team decided to exclude the staker contracts from the scope of this audit since they were still under active development at the time of the audit. No complete implementation was supplied.

Supplied in the repositories:

https://github.com/UniTradeApp/unitrade-contracts/tree/security-audit

Update: An updated version was supplied in the master branch of the repository:

https://github.com/UniTradeApp/unitrade-contracts

Notes

The audit was based on commit `69d97cd7b7ebbf7dcb0c1d392b005edc131c687a`.

Fixes: `26280c68c9f8c89d3b22fad1eeb5826e0f698d1a`

Intended Behavior

The smart contracts provide a trading layer on top of Uniswap v2, which implements an order book that allows trade to be scheduled at specified target prices.

SOLIDIFIED

Audit Report for Unitrade on October 6, 2020

Executive Summary

Solidified found that the Unitrade contracts contain 3 issues and 1 informational notes.

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

UPDATE: FIXES TO THE ISSUES HAVE BEEN SUPPLIED BY THE TEAM

Issues found:

Critical	Major	Minor	Notes
1(Fixed)	1(Acknowledged)	1(Fixed)	1

SOLIDIFIED

Audit Report for Unitrade on October 6, 2020

Issues Found

Critical Issues

1. UniTradeOrderBook.sol: Order maker or executor can drain funds contract - STATUS: FIXED

The function `updateOrder()` breaks the check-effects-interactions pattern and performs an external call before the necessary changes to the contract states are made.

A malicious maker could reenter the contract at line 150 and drain the contract's reserve of ETH. it's possible because, at this point, the totalEthDeposited hasn't been updated yet, which could be exploited by a reentrant call.

The exact same situation could be performed by the executor when receiving a fee in ETH.

Similarly, it's also possible to drain tokens that trigger execution on receive, such as ERC777, using the same method.

Recommendation
Do the value transfers calls after updating the storage variable (line 171), as relying only on the memory update wouldn't solve it.

Update
The team has fixed the issue by using Openzeppelin's `ReentrencyGuard`.

Major Issues

2. UniTradeOrderBook.sol: Potential problems with some ERC tokens - STATUS: ACKNOWLEDGED

Uniswap V2 main functions are incompatible with some types of tokens, like deflationary tokens or the ones that charge fees on transfer functions. To fix this issue, they added extra functions like `swapExactTokensForTokensSupportingFeeOnTransferTokens`.

Given the close integration with Unisawp V2, this issue also applies to the Unitrade `executeOrder` function.

In a similar vein, beware of malicious token implementations, which adhere to the ERC20 interface but do not implement it as expected.

Recommendation
This is a difficult problem to solve and it's a commonly exploited weakness of Uniswap. There's no clear recommendation on how to solve this issue at the smart contract level. We have chosen to report it in this audit, so the Unitrade team can be aware of it and make a decision considering its trade-offs. Users should be made aware of the risk of malicious tokens and imitation pairs and UI-level checks could be added.

Update
The team acknowledges the issue, which is intrinsically linked to the underlying design of Uniswap. The team proposes to mitigate the issue at the UI level, warning the user of the risk involved with ERC-20 tokens that can be added by anyone to a decentralized protocol.

Minor Issues

3. Integer Arithmetic Precision Error - STATUS: FIXED

In line 256 of `UniTradeOrderBook.sol` the following calculation is performed to calculate the amount of tokens to be burned:

```
uint burnAmount = unitradeFee.div(10).mul(6);
```

In this case, the division is performed before the multiplication, which means that precision is lost due to the remainder of the division being discarded.

Recommendation
Change the order of operations to perform multiplications before divisions.

Update: Fixed

Notes

4. Consider resetting active order index

UniTradeOrderBook.sol L235 - L251: While proceeding the order to the status `canceled` or `executed`, the `activeOrderIndex` still keeps the previous pointer. It is recommended to update the value (eg: with out of bound values like `UINT256_MAX`) for better readability later.

Disclaimer

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