Integral A-2.1

Security Audit

Jan 28th, 2022

Version 1.0.0

Prepared by

Optilistic

Introduction 3

Overall Assessment 3

Specification 3

Source Code 4

Methodology 6

Issues Descriptions and Recommendations 7

- **Severity Level Reference** 8
- [H-01] TwapPair contracts can be drained when trading is open to everyone 9
- [M-01] Gas refunds can be claimed twice 9
- [L-01] Allowance by sig 10
- [L-02] Pair id clashes 10
- [Q-01] Fee storage variables can be compacted 11
- [Q-02] Unnecessary struct wrapping 11

Automated Analysis 12

- Slither 12

Diagram A 13

Disclaimer 14

Introduction

This document includes the results of the preliminary security audit for a subset of Integral's smart contract code as found in the section titled 'Source Code'. The security audit was performed by the Optilistic team from November 24, 2021 to December 22, 2021.

The purpose of this audit is to review the source code of certain Integral Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Optilistic's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

We identified a few issues of low to high severity. Integral was quick to respond and fix these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Slack with the Integral team
- The Integral docs website

Source Code

The following source code was reviewed during the audit:

Repository	Commit
Github	c422533e5deefc43d67533072b86dac407a1e6ec

Specifically, we audited the following contracts:

Contract	Sha256
TwapDelay.sol	90ba9f245148e812972f68bff39158294abd73b67a3b7e10210630013f50b95f
TwapFactory.sol	13806ee163cb31e148e7ddf49d99334304c29f8a7ecae6c01168f6ff245e4c04
TwapLPToken.sol	70eac8396ba70321c65e0dca82df4d7e2f001d8c23f6f39b628647bec1dfa9d1
TwapOracle.sol	300c96af631a94bbc1e5590c33b30d91ceb9c4be4656610f5eba1b9db920d9f5
TwapPair.sol	89111682842d3bf40d8efb1a3c1e5cab298968c0d0ffbc7f3165cbfbaac3fe8f
TwapReader.sol	78000c0b7e269eecba616ae9e97f0d641ed9b54a8efb93201f1e6601bc7f6d07
interfaces/IERC20.sol	77254a6e6f31346f157ebaf277dee676b588d69738d30bf8313d28403c9b07f1
interfaces/IReserves.sol	51230757f347e74db1fdc073ce6e991cc999311b6e55223f784dcce55540529b
interfaces/ITwapDelay.sol	3567a299aa561414a47ba00f260a16b0fd55126ef3ebd898ffe0746bfd691269
interfaces/ITwapERC20.sol	500933ee5eaabc9a68276e262ddafd71568a17e1b6e672e740dc2b76ed7270a9
interfaces/ITwapFactory.sol	e2221f5cbde9efa9526e48435a022d6238c8064225ff2fe1f5fb71f376e8392e
interfaces/ITwapMath.sol	0df9cbcc3d0c7caec97881c42acfdf08b671e4447e3332d14a46b68d4ab7beda

interfaces/ITwapOracle.sol	8a7406b6fa76f4042178ad01684e4c9c7ea669b1e2fe03d8f4492946f0519c6c
interfaces/ITwapPair.sol	15df16fedf229080711bf0d0ad116f8198f3f13bb3e7218e63c05f9c055e9ec9
interfaces/ITwapReader.sol	052f4e29694d5488808b11444de6d68a06fff1036d4918fa32054325c7b66d1b
interfaces/IWETH.sol	451b984799684cd7f42594b93e8594a0655f7444e627d82b235a262ca17207c8
libraries/AbstractERC20.sol	7f421562b933675ecb1fbc7e81f28b906505abac379c368499129efc651ff21a
libraries/AddLiquidity.sol	0c37bc5ff9f28237e8b84623e8c458f23ae17467277dc8e6569d9e7797605c3d
libraries/Math.sol	ab329e9068ad63e37da4e59ba3406050507b83221c301f08e3744c371567ad6a
libraries/Orders.sol	5ed0e6ca3da546d1f0d662ecc6136fb4aaf94bce3bedd08bd0fb725d835b3b28
libraries/Reserves.sol	ed8311d011d3e30ebe8d5491f45d0126a7d1d8c796a45f049fb57c6e5ed87a46
libraries/SafeMath.sol	59a4e34799a06e010557e9c8b77385e8433da831102b807b1d8290b72639ea59
libraries/TokenShares.sol	aba3b40c350d775a9aa438f66a4f153f1ef3ab6e97925f152a0c908eb1377bed
libraries/TransferHelper.sol	0023d01c5338c77dc5d1862adf1abb7c49ce2f192278515760a7b02624d82859
libraries/WithdrawHelper.sol	4ee715d0764c98bc0a6990ad0b89dce954eb444533ea646f6f8f8e0fe40874c8

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Methodology

The audit was conducted in several steps.

First, we reviewed in detail all available documentation and specifications for the project, as described in the 'Specification' section above.

Second, we performed a thorough manual review of the code, checking that the code matched up with the specification, as well as the spirit of the contract (i.e. the intended behavior). During this manual review portion of the audit we primarily searched for security vulnerabilities, unwanted behavior vulnerabilities, and problems with systems of incentives.

Third, we performed the automated portion of the review consisting of measuring test coverage (while also assessing the quality of the test suite) and evaluating the results of various symbolic execution tools against the code.

Lastly, we performed a final line-by-line inspection of the code – including comments –in effort to find any minor issues with code quality, documentation, or best practices.

Issues Descriptions and Recommendations

Issues Descriptions and Recommendations **7**

Severity Level Reference 8

[H-01] TwapPair contracts can be drained when trading is open to everyone 9

[M-01] Gas refunds can be claimed twice 9

[L-01] Allowance by sig 10

[L-02] Pair id clashes 10

[Q-01] Fee storage variables can be compacted 11

[Q-02] Unnecessary struct wrapping 11

Automated Analysis **12**

Slither 12

Diagram A **13**

Disclaimer **14**

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

[H-01] TwapPair contracts can be drained when trading is open to everyone

~~HIGH~~ *Fixed by f1f998aa5dcca71a7dd5a40b625adcf0be411815*

`TwapPair.swap()` takes an arbitrary `bytes calldata data` parameter and passes it to `TwapOracle.tradeX/tradeY` with no validation. TwapOracle decodes this parameter to use as the price to trade at.

The `trader` variable can be set to `address(-1)`, which allows anyone to call `swap()` directly, according to `canTrade()`.

Calling the swap function directly, this behavior allows an attacker to specify a price that is much cheaper than the current reserve ratio and make trades at that price, potentially draining all funds from the pool across multiple trades.

Consider pulling price data from a trusted source, or removing the open access feature from TwapPair.

[M-01] Gas refunds can be claimed twice

~~MEDIUM~~ *Fixed by 2f8cc18ca7de126d35a2082fb35b507b9e2da549*

An exploit contract can create a sure-failing buy order and reject the token refund but accept the gas refund. Later, the exploit contract calls `cancelOrder` on that order and receives a second gas refund (in addition to accepting the original token refund). See Diagram A for more details.

This issue is mitigated by several factors, such as Orders.Data.maxGasLimit and the team's ability to pause new orders.

Consider restricting cancellations to orders that have not been executed.

[L-01] Allowance by sig

LOW

`increaseAllowance` and `decreaseAllowance` exist to protect against front-running attacks. However, `permit` only behaves as an equivalent to `approve`.

Consider also providing a way to increase/decrease allowance by sig.

[L-02] Pair id clashes

~~LOW~~ *Addressed by https://github.com/EmergentHQ/integral/pull/1077*

Orders.sol generates a `pairId` taking the k-hash of a TwapPair's address. While it is highly unlikely that two pair contracts will conflict, it is still possible (similar to function selectors), and would cause user funds to be stuck inside TwapDelay. To illustrate:

- Assume X and Y are tokens, and the pairId's for ETH-X and ETH-Y clash.
- ETH-X is created and used first, thus setting its Orders.Data.pairs key.
- ETH-Y is created later.
    a. Although there is a clash, no error occurs. See Orders.sol:243
- User deposits into ETH-Y, transferring their Y to TwapDelay via tokenShares.amountToShares
    a. The clashing pairId is stored in the order
- Later, `_initialDeposits` is run for ETH-X instead of ETH-Y.
- At this point, two options are possible:
    a. The contract deposits someone else's X funds, which will cause problems for that other user later.
    b. The contract has no X funds to deposit, in which case the deposit reverts, causing TwapDelay to attempt to refund ETH-X, which will itself fail.

This is only a low-severity issue because creating pairs is restricted to the TwapFactory owner. However, this becomes a high severity problem if pair creation is opened up to the public in some way.

Options to consider:

- Move the pairs mapping to the TwapFactory contract and check for clashes in createPair
- Being very careful when creating new pairs (only works if pair creation is always restricted to admins)

[Q-01] Fee storage variables can be compacted

~~GODE QUALITY~~ *Resolved by https://github.com/EmergentHQ/integral/pull/1080*

`fee0` and `fee1` are both declared as `uint256`.

Consider using `uint112` to save storage space, as well as match the reserve variable sizes.

[Q-02] Unnecessary struct wrapping

~~GODE QUALITY~~ *Resolved by https://github.com/EmergentHQ/integral/pull/1090*

In TwapOracle's `getAveragePrice` function, a `FixedPoint.uq112x112` struct is initialized, but only to be unwrapped afterwards.

Consider removing the use of this struct.

Automated Analysis

Slither

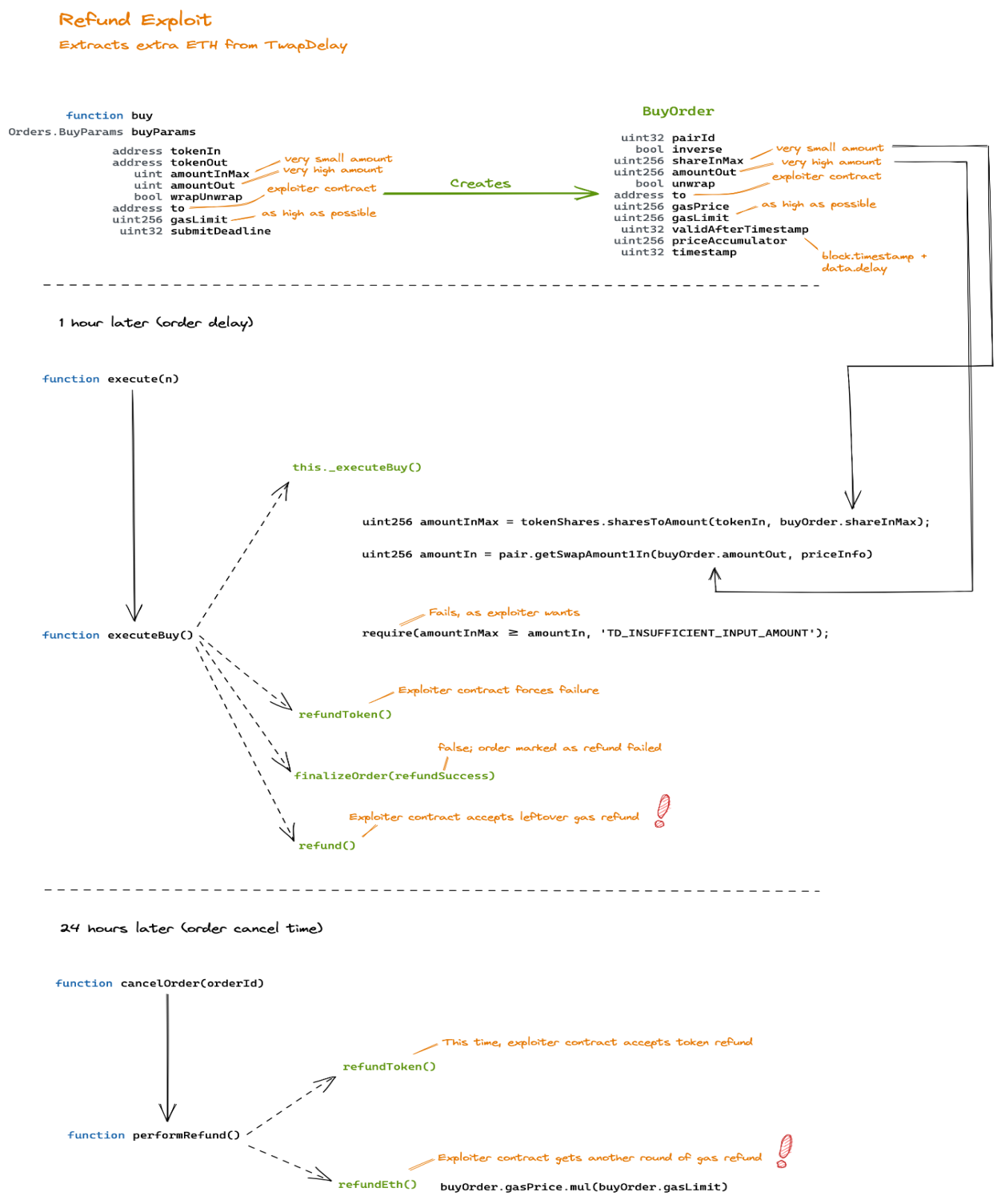
Slither is a solidity static analysis framework. It detects many vulnerabilities, from high threats to benign ones, of which there are usually many.

In order to run Slither against the codebase we ran the following command and filtered for relevant files:

- `$ slither .`

Slither identified some benign reentrancy vulnerabilities, but manual inspection revealed them to be false positives.

[M-01] Diagram A



Disclaimer

Optilistic makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Optilistic specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Optilistic will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Optilistic be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Optilistic has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Optilistic notes as being within the scope of Optilistic's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Optilistic. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Optilistic is not responsible for the content or operation of such websites, and that Optilistic shall have no liability to your or any other person or entity for the use of third party websites. Optilistic assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.