0xSplits A-1

Security Audit

Feb 14th, 2022

Version 1.0.0

Prepared by

Optilistic

Optilistic

Introduction

This document includes the results of the security audit for 0xSplit's smart contract code as found in the section titled 'Source Code'. The security audit was performed by the Optilistic team from Jan 24, 2022 to Feb 7, 2021.

The purpose of this audit is to review the source code of certain 0xSplits 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 non-severe to medium severity. 0xSplits was quick to respond and fix these issues.

Specification

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

- Discussions on Discord with the 0xSplits team.
- The official website and docs.

Source Code

The following source code was reviewed during the audit:

Repository	Commit
Github	eb909d91fb4b5b94123e36f0b927b9997c67653a

Specifically, we audited the following contracts:

Contract	Sha256
Multicall2.sol	5218c1400f46b1a176fe5d5a65686b2b6e8be1845e4de178920c4a7bfeecca8a
SplitMain.sol	c265420ecbb7314ff334817a8ffef4a9fecf118e8c51422a6bd93eae3e960275
SplitWallet.sol	a942d6c8046e23dbd398b2f225bd827e4f2e8abd9660dc62e281a5eeae84a609
interfaces/ISplitMain.sol	db133e8f822cdd302b9cb8d752fa9eaeb26b5ebde38ea61219b39ed232e00df1
interfaces/ReverseRecords.sol	9f61c53da833289efe01e58fdbbe45e7e43fa61c343a758b903c051e3167fae4
libraries/Clones.sol	a9c796d734deccbe6d7ca2d24cd58da319fa883e83f2698fea2661e36387d3b0

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

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.

[M-01] Tax-on-transfer ERC-20 Token Issue

MEDIUM

SplitMain.sol's `distributeERC20` initiates a ERC-20 transfer from a SplitWallet to itself:

```
uint256 proxyBalance = token.balanceOf(split);
// …
SplitWallet(split).sendERC20ToMain(token, proxyBalance - 1);
```

This code assumes that `proxyBalance` is the final amount that actually got transferred. However, this results in a discrepancy when transferring a "tax-on-tranfer" token – a token that transfers a full amount from the sender, but a smaller amount to the receiver.

This discrepancy causes SplitMain to think it has a higher balance than it does, which leads to the last withdrawing recipient unable to withdraw.

Consider reading from SplitMain's token balance before and after the transfer from SplitWallet, and then using the difference in `amountToSplit`.

[Q-01] Name clarification

~~CODE QUALITY~~ *Fixed by 58dabee67c236f94bd23637e22cd6108782a87b3*

Consider naming `predictSplitAddress` something like `getImmutableSplitAddress` to explicitly associate the behavior to immutable splits (and avoid any confusion with mutable split addresses).

[Q-02] `distributeETH` and `distributeERC20` Gas optimization

~~CODE QUALITY~~ *Fixed by f64ccd653eab6c3df1004a42f527196200a3afbf*

In SplitMain.sol, line 394 always sets `ethBalances[split]` to the value `1` (and similarly on line 478). However, the only way the original value can be greater than one is if this split is the recipient of another split, which is relatively rare.

Updating the condition to be greater than zero (and moving some math logic around) instead of `!= 1` to save the ~20k storage write gas cost until it's necessary.

[Q-03] Withdrawal event refactor

~~CODE QUALITY~~ *Fixed by 2a417645a62855bd78b65b2a1e2628801f1bb183*

SplitMain's `withdraw` function emits an event, which contains all the successfully withdrawn amounts. However, the semantic type of the emitted `withdrawnAmounts` is dependent on whether `eth` is true or false.

Consider having `withdrawnAmounts`'s length always correspond to `tokens`, and emitting the withdrawn amount of ETH instead of the `eth` boolean, to make the semantic type of `withdrawnAmounts` consistent.

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 "ether locking" vulnerabilities; manual inspection revealed them to be false positives.

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.