Impossible Finance

SMART CONTRACT AUDIT

ZOKYO.

June 14th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

SCORE

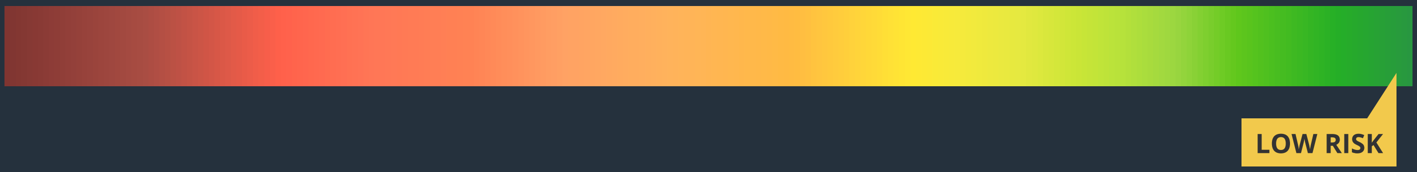
97

TECHNICAL SUMMARY

This document outlines the overall security of the Impossible Finance Swap smart contracts, evaluated by Zokyo's Blockchain Security team.

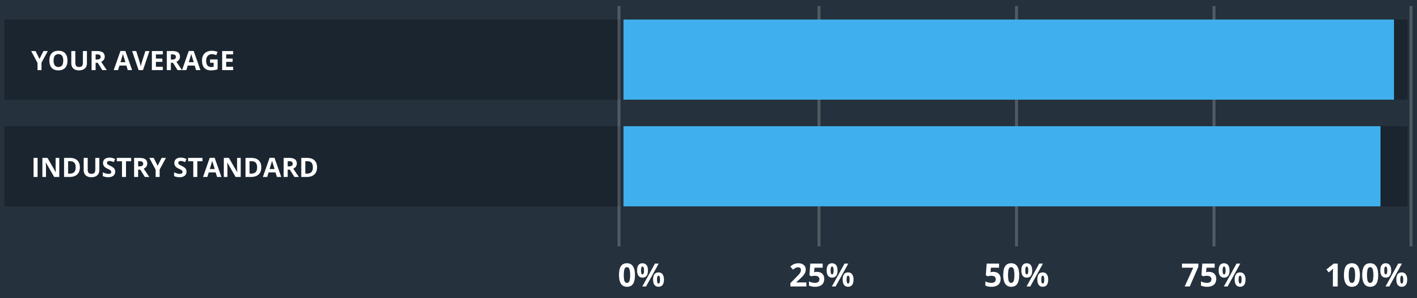
The scope of this audit was to analyze and document the Impossible Finance smart contract codebase for quality, security, and correctness.

Contract Status



There were no critical issues found during the audit.

Testable Code



The testable code is 97%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Impossible Finance team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Impossible Finance - Impossible Swap repository.

Repository - https://github.com/ImpossibleFinance/impossibleSwapCore
Commit id - 3bc771b904d7c64f61a4afe88f4d6acdaa39bcd3

Last commit reviewed - a3208bcf64141bad4ca6b7bf05989943cf68453c

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):
ImpossibleERC20.sol
ImpossibleFactory.sol
ImpossiblePair.sol
ImpossibleRouter01.sol
ImpossibleRouter02.sol
ImpossibleLibrary.sol

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Impossible Finance smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1 Due diligence in assessing the overall code quality of the codebase.

3 Testing contract logic against common and uncommon attack vectors.

2 Cross-comparison with other, similar smart contracts by industry leaders.

4 Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were no critical issues found during the audit. Though there was found incorrect calculation, All other mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

The findings during the audit have a slight impact on contract performance or security, and can affect the further development.

Nevertheless, all findings were successfully fixed by the Impossible Finance team.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

● High

The issue affects the ability of the contract to compile or operate in a significant way.

● Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

● Low

The issue has minimal impact on the contract’s ability to operate.

● Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | RESOLVED

Error in fee calculation.

ImpossibleLibrary.sol, Line 112
amountInPostFee = amountInPostFee.sub(sqrtK.sub(reserveIn));
- amountInPostFee is powered by 10000
- sqrtK.sub(reserveIn) - regular.
There is a calculation mistake with the loose of accuracy

Recommendation:
Fix the calculation mistake

MEDIUM | RESOLVED

Different Solidity versions.

Different pragma directives are used. Throughout the project (including interfaces). Version used: '=0.5.16', '>=0.5.0'
Issue is classified as Medium, because it is included to the list of standard smart contracts' vulnerabilities.

Recommendation:
Use the same pragma directives for the entire project.

MEDIUM | RESOLVED

Solidity version update.

The solidity version should be updated. Throughout the project (including interfaces). Issue is classified as Medium, because it is included to the list of standard smart contracts' vulnerabilities.

Recommendation:
You need to update the solidity version to the latest one - at least to 0.6.12, though 0.7.6 will be the best option. This will help to get rid of bugs in the older versions.

MEDIUM | RESOLVED

Hardcoded address

StableXPair.sol, line 189
The governance address has unique permissions to change the setter and collect the fees. Though there is no way to change that address and the address is hardcoded, so the contract has no ability to change that permission in case of governance change or compromise.

Recommendation:
Add the governance address to the arguments of the initialize function and/or add the ability in the StableXFactory.sol contract to change this address and pass the required one.

LOW | RESOLVED

Explicitly mark visibility of state

StableXPair.sol The variables do not specify the visibility of the state. lines 34-42, 45, 46.

Recommendation:
Explicitly mark visibility of state.

LOW | RESOLVED

Functions should be declared as external

ImpossiblePair.getFeeAndXybk() (ImpossiblePair.sol#67-70)
ImpossiblePair.updateGovernance(address) (ImpossiblePair.sol#140-143)
ImpossiblePair.makeXybk(uint8,uint8,uint32,uint32) (ImpossiblePair.sol#146-169)
ImpossiblePair.makeUni() (ImpossiblePair.sol#173-183)
ImpossiblePair.updateTradeFees(uint16) (ImpossiblePair.sol#185-190)
ImpossiblePair.updateDelay(uint256) (ImpossiblePair.sol#194-198)
ImpossiblePair.updateHardstops(uint8,uint8) (ImpossiblePair.sol#201-207)
ImpossiblePair.updateBoost(uint32,uint32) (ImpossiblePair.sol#210-221)
ImpossibleRouter01.quote(uint256,uint256,uint256) (ImpossibleRouter01.sol#344-350)
ImpossibleRouter02.quote(uint256,uint256,uint256) (ImpossibleRouter02.sol#470-476)
ImpossibleRouter01.getAmountOut(uint256,address,address)
(ImpossibleRouter01.sol#352-358)
ImpossibleRouter02.getAmountOut(uint256,address,address)
(ImpossibleRouter02.sol#478-484)
ImpossibleRouter01.getAmountIn(uint256,address,address)
(ImpossibleRouter01.sol#360-366)
ImpossibleRouter02.getAmountIn(uint256,address,address)
(ImpossibleRouter02.sol#486-492)
ImpossibleRouter01.getAmountsOut(uint256,address[]) (ImpossibleRouter01.sol#369-376)
ImpossibleRouter02.getAmountsOut(uint256,address[]) (ImpossibleRouter02.sol#494-502)
ImpossibleRouter01.getAmountsIn(uint256,address[]) (ImpossibleRouter01.sol#379-386)
ImpossibleRouter02.getAmountsIn(uint256,address[]) (ImpossibleRouter02.sol#504-512)

Recommendation:
Functions should be declared as external

LOW | RESOLVED

Extra variable

StableXPair.sol line 177, 181. It is possible to optimize the code for lower gas consumption. Immediately assign the blockTimestampLast value without using temporary variables.

Recommendation:
Set the blockTimestampLast value without using the blockTimestamp variable.

INFORMATIONAL | RESOLVED

Unused contract

Migration.sol
Standard Migration contract does not have any role in the system and can be removed.

Recommendation:
Remove the unused standard contract.

INFORMATIONAL | RESOLVED

Set the commission percentage to a variable

StableXPair.sol line 252, 253.
In order to increase the readability of the code and further development security it is recommended to move the commision value (201) to the public variable or public constant.

Recommendation:
Move the value into a constant.

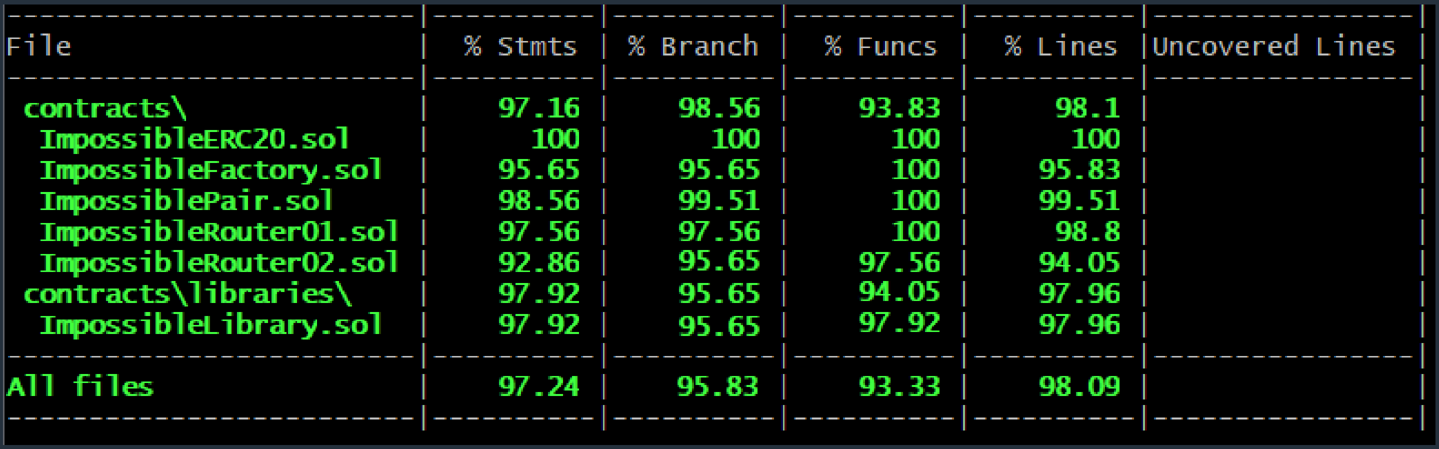
	ImpossibleFactory	ImpossibleRouter (1 and 2)	ImpossiblePair
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

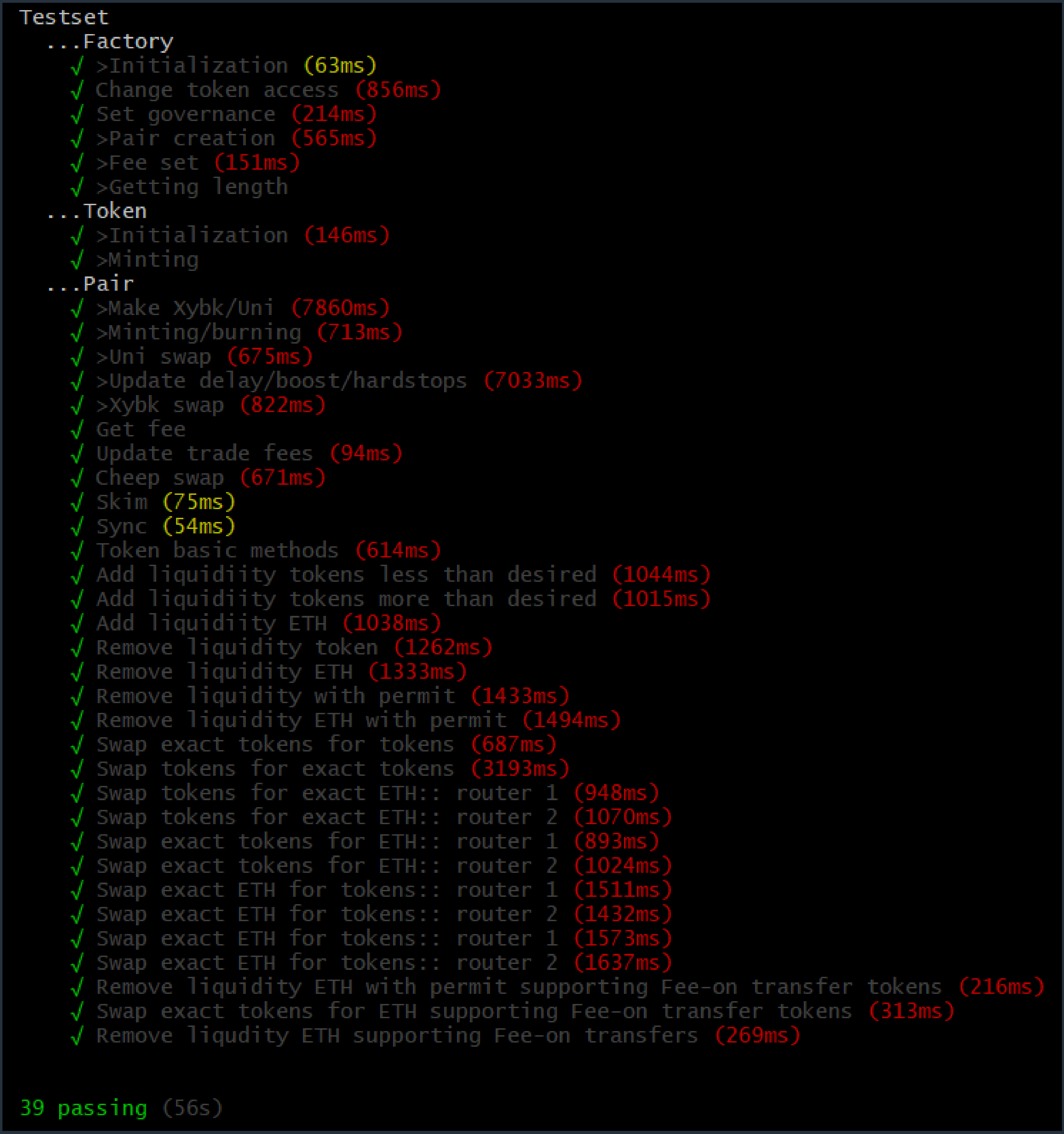
Tests written by Zokyo team

As part of our work assisting Impossible Finance in verifying the correctness of their contract code, our team was responsible for writing integration tests using Truffle testing framework.

Tests were based on the functionality of the code, as well as review of the Impossible Finance contract requirements for details about issuance amounts and how the system handles these.



```
-------------------------|----------|----------|----------|----------|----------------|
File                     |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------------|----------|----------|----------|----------|----------------|
 contracts\              |    97.16 |    98.56 |    93.83 |     98.1 |                |
  ImpossibleERC20.sol    |      100 |      100 |      100 |      100 |                |
  ImpossibleFactory.sol  |    95.65 |    95.65 |      100 |    95.83 |                |
  ImpossiblePair.sol     |    98.56 |    99.51 |      100 |    99.51 |                |
  ImpossibleRouter01.sol |    97.56 |    97.56 |      100 |     98.8 |                |
  ImpossibleRouter02.sol |    92.86 |    95.65 |    97.56 |    94.05 |                |
 contracts\libraries\    |    97.92 |    95.65 |    94.05 |    97.96 |                |
  ImpossibleLibrary.sol  |    97.92 |    95.65 |    97.92 |    97.96 |                |
-------------------------|----------|----------|----------|----------|----------------|
All files                |    97.24 |    95.83 |    93.33 |    98.09 |                |
-------------------------|----------|----------|----------|----------|----------------|
```



```
Testset
  ...Factory
    √ >Initialization (63ms)
    √ Change token access (856ms)
    √ Set governance (214ms)
    √ >Pair creation (565ms)
    √ >Fee set (151ms)
    √ >Getting length
  ...Token
    √ >Initialization (146ms)
    √ >Minting
  ...Pair
    √ >Make Xybk/Uni (7860ms)
    √ >Minting/burning (713ms)
    √ >Uni swap (675ms)
    √ >Update delay/boost/hardstops (7033ms)
    √ >Xybk swap (822ms)
    √ Get fee
    √ Update trade fees (94ms)
    √ Cheep swap (671ms)
    √ Skim (75ms)
    √ Sync (54ms)
    √ Token basic methods (614ms)
    √ Add liquidiity tokens less than desired (1044ms)
    √ Add liquidiity tokens more than desired (1015ms)
    √ Add liquidiity ETH (1038ms)
    √ Remove liquidity token (1262ms)
    √ Remove liquidity ETH (1333ms)
    √ Remove liquidity with permit (1433ms)
    √ Remove liquidity ETH with permit (1494ms)
    √ Swap exact tokens for tokens (687ms)
    √ Swap tokens for exact tokens (3193ms)
    √ Swap tokens for exact ETH:: router 1 (948ms)
    √ Swap tokens for exact ETH:: router 2 (1070ms)
    √ Swap exact tokens for ETH:: router 1 (893ms)
    √ Swap exact tokens for ETH:: router 2 (1024ms)
    √ Swap exact ETH for tokens:: router 1 (1511ms)
    √ Swap exact ETH for tokens:: router 2 (1432ms)
    √ Swap exact ETH for tokens:: router 1 (1573ms)
    √ Swap exact ETH for tokens:: router 2 (1637ms)
    √ Remove liquidity ETH with permit supporting Fee-on transfer tokens (216ms)
    √ Swap exact tokens for ETH supporting Fee-on transfer tokens (313ms)
    √ Remove liqudity ETH supporting Fee-on transfers (269ms)


39 passing (56s)
```

We are grateful to have been given the opportunity to work with the Impossible Finance team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Impossible Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.