ackee blockchain

1inch Fixed Rate Swap Audit

22. 8. 2021

by Ackee Blockchain

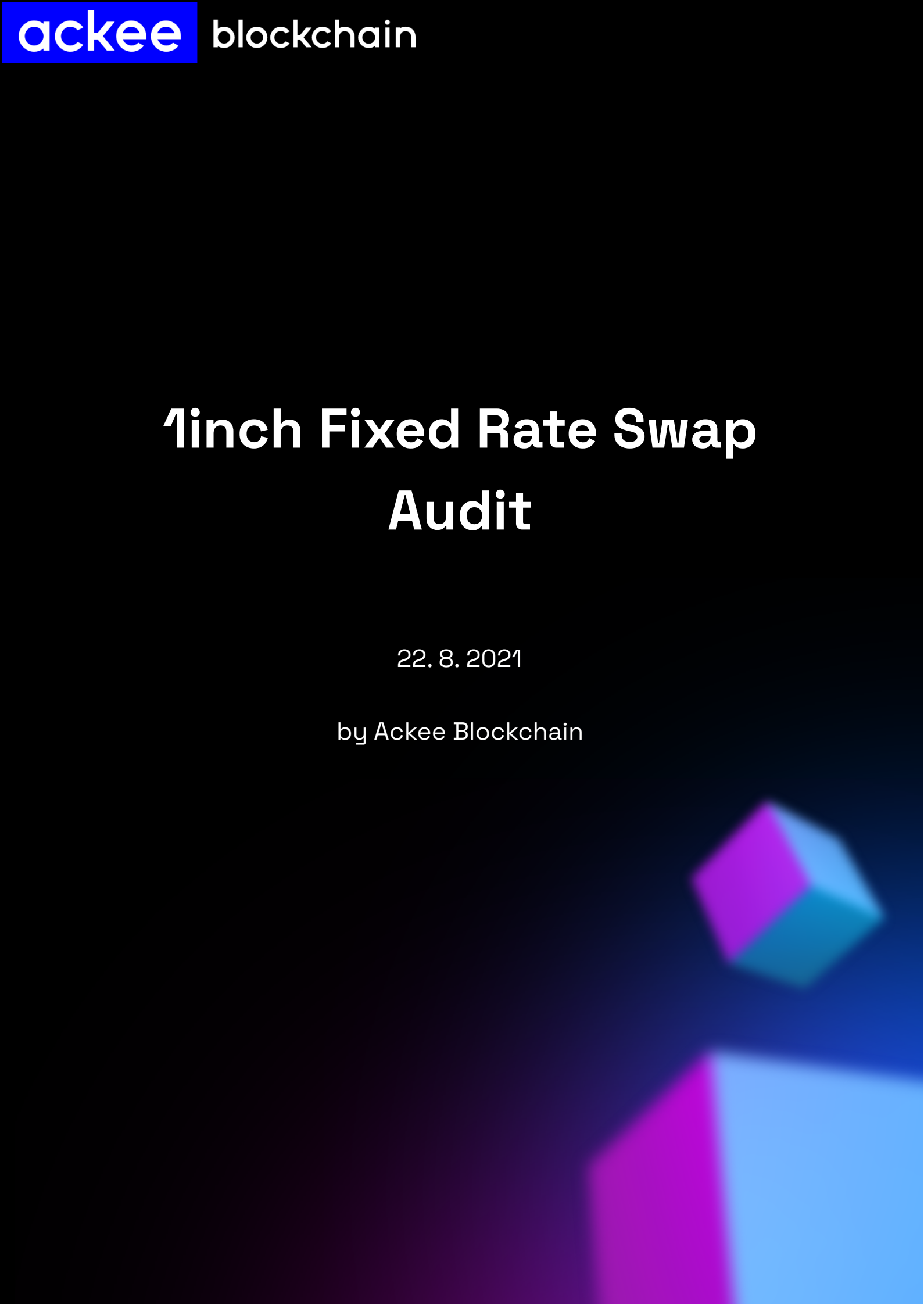


Table of Contents

1. Overview

This document presents our findings in a reviewed contract.

1.1 Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

1.2 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - basic check with automated Solidity analysis tools MythX and Slither is performed.
3. **Math validation** - mathematical calculations in the code are manually validated iif results behave as defined.
4. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
5. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
6. **Unit testing and fuzzy testing** - additional unit tests are written in the Brownie testing framework to ensure that the system works as expected. Fuzzy testing is performed by Echidna.

1.3 Review team

The audit has been performed with a total time donation of 3 engineering days. The work was divided between the Chief Auditor who defined the methodology and performed manual code review. Automated tests were implemented by a tester on the Chief Auditor's assignment. The whole process was supervised by the Audit Guarantee.

Member's Name	Position
Stepan Sonsky	Chief Auditor
Tester 1	Tester
Josef Gattermayer, Ph.D.	Audit Guarantee

1.4 Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues.

2. Scope

This chapter describes the audit scope, contains provided specification, used documentation and set main objectives for the audit process.

2.1 Coverage

Files being audited:

- FixedRateSwap.sol (154 SLOC)

Sources revision used during the whole auditing process:

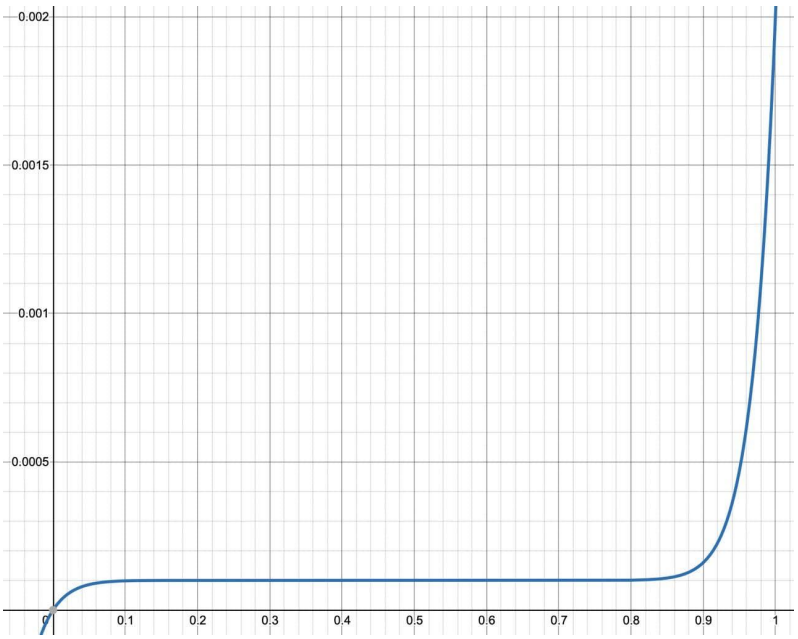
- Repository: https://github.com/1inch/fixed-rate-swap
- Commit: 60a36947261bfe8e2914684f74c1ca72060cf3e3

2.2 Supporting Documentation

1inch developers provided a brief description how the system should behave and how are the fees calculated:

"*FixedRateSwap is a simple AMM that allows swapping tokens with 1:1 rate and variable fee. Deposit method is limited to onlyOwner by design. Fee is calculated depending on the ratio of balances of the tokens in the contract. The logic of choosing the constants*

getReturn(x) = 0.9999 + (ax + b)^k
when x = 0 => f(x) = 1 => b^k = 0.0001
when x = 1 => f(x) = 0.998 => (a + b)^k = -0.0019 => k should be odd
let's take k = 17, then
b = 0.5817091329374359
a = -1.2734233188154198



This is the chart of 1 - getReturn(x) which is equal to fee.

It is almost always 0.01% but drops down to 0% when there is a liquidity imbalance and the user tries to fix it, while it also rises to 0.2% when there is a liquidity imbalance and the user tries to worsen it.

x is the ratio of inputBalance / (inputBalance + outputBalance)

And then to calculate actual fee we use integration over the segment [inputBalance / (inputBalance + outputBalance), (inputBalance + inputAmount) / (inputbalance + outputBalance)]"

Used OpenZeppelin documentation:

- https://docs.openzeppelin.com/contracts/4.x/api/token/erc20
- https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20
- https://docs.openzeppelin.com/contracts/4.x/api/access#Ownable
- https://docs.openzeppelin.com/contracts/4.x/api/utils#math

2.3 Objectives

We've defined following main objectives of the audit:

- Check the code quality, architecture and best practices.
- We should double check if mathematical algorithms are working as described, so there is no possibility of losing the funds due to mathematical error.
- Also it's important to ensure that nobody unauthorized can steal the funds held by the contract.
- Since the contract doesn't use a proxy upgrade pattern, we also need to focus on potential denial of service attacks.

3. System Overview

This chapter describes the audited system from our understanding.

Project contains 1 smart contract, which is based on the ERC20 standard from the OpenZeppelin library and uses 2 other tokens to swap. FixedRateSwap is an AMM (Automated Market Maker) to provide liquidity for swaps with 1:1 rate and variable fees. Only the owner can deposit tokens and provide liquidity.

3.1 FixedRateSwap.sol

Following chapter describes our detailed understanding of FixedRateSwap.sol and its parts.

Dependencies

`@openzeppelin/contracts/access/`**`Ownable.sol`**

- Module which provides a basic access control mechanism. Used for transfer an ownership of the contract.

`@openzeppelin/contracts/utils/math/`**`Math.sol`**

- Math utilities, used in the project for min(uint256 a, uint256 b) function.

`@openzeppelin/contracts/token/ERC20/`**`ERC20.sol`**

- ERC20 token standard implementation. Used as parent of FixedRateSwap.

`@openzeppelin/contracts/token/ERC20/utils/`**`SafeERC20.sol`**

- Wrapper around the interface that eliminates the need to handle boolean return values.

Constants

```
uint256 constant private _ONE = 1e18
uint256 constant private _C1 = 0.9999e18;
uint256 constant private _C2 = 3.3827123349983306e18
uint256 constant private _C3 = 0.4568073509746632e18
```

Constant _ONE is used to calculate precision math, because Solidity lacks decimal number types. _C1, _C2 and _C3 comes from equations to achieve the desired fee curve.

Variables

```
IERC20 immutable public token0
IERC20 immutable public token1
uint8 immutable private _decimals
```

In token0 and token1 variables are saved addresses of tokens used to swap. Variable _decimals holds the count of decimals of the liquidity provider token.

Functions

```
constructor(IERC20 _token0, IERC20 _token1, string memory name,
string memory symbol, uint8 decimals_)
```

- Initializes the contract with both token addresses, name, symbol and decimals count, calls super ERC20 constructor and saves values into immutable state variables.

```
decimals() public view virtual override returns(uint8)
```

- Overriding function of ERC20, returns number of LP token decimals.

```
getReturn(IERC20 tokenFrom, IERC20 tokenTo, uint256 inputAmount)
public view returns(uint256 outputAmount)
```

- Calculates outputAmount of destination token based on inputAmout of source token. Regarding our simulations, this function works as described in provided documentation.

```
deposit(uint256 token0Amount, uint256 token1Amount) external
returns(uint256 share)
```

- Deposits token amounts into the contract, calls depositFor() with msg.sender param.

```
depositFor(uint256 token0Amount, uint256 token1Amount, address to)
public onlyOwner returns(uint256 share)
```

- Deposits token0 and token1 amounts into the contract. Calculates share and mints that amount of LP token to msg.sender. This method is restricted to be called only by the contract owner, so no one else can provide liquidity.

```
withdraw(uint256 amount) external returns(uint256 token0Share,
uint256 token1Share)
```

- Withdraws tokens from the contract, calls withdrawFor() with msg.sender param.

withdrawFor(uint256 amount, address to) public returns(uint256 token0Share, uint256 token1Share)

- Calculates the token0 and token1 share for provided LP amount, burns that amount and withdraws appropriate token0 and token1 shares to provided address.

swap0To1(uint256 inputAmount) external returns(uint256 outputAmount)

- Swaps inputAmount of token0 to token1. Calls _swap function.

swap1To0(uint256 inputAmount) external returns(uint256 outputAmount)

- Swaps inputAmount of token1 to token0. Calls _swap function.

swap0To1For(uint256 inputAmount, address to) public returns(uint256 outputAmount)

- Swaps inputAmount of token0 from msg.sender to token1 and transfers it to address passed by argument. Calls _swap function.

swap1To0For(uint256 inputAmount, address to) public returns(uint256 outputAmount)

- Swaps inputAmount of token1 from msg.sender to token0 and transfers it to address passed by argument. Calls _swap function.

_swap(IERC20 tokenFrom, IERC20 tokenTo, uint256 inputAmount, address to) private returns(uint256 outputAmount)

- Calculates outputAmount from inputAmout using getReturn() function. Transfers inputAmout of tokenFrom from msg.sender to the contract and transfers outputAmount of tokenTo from the contract to address passed by argument.

_powerHelper(uint256 x) private pure returns(uint256 p)

- Helper function for calculating ^18 of the number p, which depends on condition (x > _C3). Function is dividing intermediate results by _ONE to prevent overflow.

4.Security Specification

This section specifies single roles and their relationships in terms of security in our understanding of the audited system. These understandings are later validated.

4.1 Actors

This part describes actors of the system, their roles and permissions.

Owner + Liquidity Provider

The owner account deploys the contracts to the network. Ownership can be transferred to another account by the current owner. Only the owner can deposit tokens into the contract. Regarding the developer's answer, an owner account is secured using foundation multisig.

Trader

The trader could be any user of the network. Can swap token0 to token1 and vice-versa. Also users are able to withdraw token0 and token1 for respective LP token amount.

4.2 Trust model

Since traders can't deposit tokens into the contract, potential theft can affect only the owner's funds. Users have to trust developers that math algorithms are correctly implemented and they receive the correct amount of token1 for token0 and vice-versa.

5. Findings

This chapter shows detailed output of our analysis and testing.

5.1 General Comments

The overall code quality is good. Functions are well designed to avoid code duplicity. Executions of functions with invalid parameters are properly handled using require. However the code isn't well documented, only the getReturn() function is commented, so we highly recommend covering all of the functions with documentation. Regarding our simulations, swap fee policy behaves like it's described in chapter 2.2.

5.2 Issues

Using our toolset, manual code review, unit testing and fuzzy testing we've identified the following issues.

Low

Low severity issues are more comments and recommendations rather than security issues. We provide hints on how to improve code readability and follow best practices. Further actions depend on the development team decision.

ID	Description	Contract	Line
L1	SWC-103 Floating pragma	FixedRateSwap.sol	3
L2	Code duplicity	FixedRateSwap.sol	56 57 58

L1: This issue was generated from automated tools and doesn't follow Solidity best practices, also known as SWC-103. It's recommended to lock pragma to a specific compiler version.

L2: (fromBalance + toBalance) should be assigned to a local variable to improve code readability and it also saves a little bit of gas.

Medium

Medium severity issues aren't security vulnerabilities, but should be clearly clarified or fixed.

ID	Description	Contract	Line
M1	Potential token decimals mismatch	FixedRateSwap.sol	25 26
M2	Unhandled division by zero - Zero and negative inputAmount is not handled before math operations.	FixedRateSwap.sol	63

M1: Presents a hypothetical problem in stablecoins with different decimals. If it's a potential use case, please pay attention to the following scenario, how to reproduce the issue:

1. Create two stablecoins, e.g. USDX with 2 decimals and USDY with 4 decimals.
2. Deploy FixedRateSwap with these tokens and deposit equal value of both tokens to the contract, let's say value of $10000, so it's 10000|00 USDX and 10000|0000 USDY. (Character | is imaginary decimal point for better understanding).
3. Now swap value of $1000 from USDX to USDY, so 1000|00 USDX.
4. You'll receive approx. 9|9999 USDY, which means a value of cca $9.99.

So we propose 3 possible solutions to this issue:

1. This case is not relevant and the situation cannot happen during the real contract usage => No action needed.
2. Add require(_token0.decimals() == _token1.decimals(), "Decimals mismatch") to the constructor to ensure decimals equality during the deployment.
3. Update mathematical operations to support different decimals and behave correctly in described scenarios.

M2: Zero and negative inputAmount is not handled before math operations. We recommend adding require(inputAmount > 0, "Invalid input amount") at the beginning of getReturn() method. See more in Appendix A.

High

High severity issues are security vulnerabilities, which require specific steps and conditions to be exploited. These issues have to be fixed.

ID	Description	Contract	Line
H1	Unauthorized withdrawal	FixedRateSwap.sol	93 97

H1: We've identified a potential risk of an unauthorized withdrawal. If an attacker somehow seize LP tokens, he would be able to withdraw token0 and token1. It's not a directly exploitable issue and requires ownership of LP tokens as a prerequisite, however it isn't an impossible scenario, even though we haven't found the direct exploit. We propose adding onlyOwner modifier also to the withdrawFor() function for additional layer of security.

Critical

Direct critical security threats, which could be instantly misused to attack the system. These issues have to be fixed.

✓ We haven't found any critical severity issues.

5.3 Unit testing

Mostly revert scenarios are uncovered and it's probably the reason why for example M2 was undiscovered. In addition, tests for the rest of public functions (not only external), could be potentially useful.

Overview

ID	Description	Contract/Function	Result
U1	Revert: "Deposit to this is forbidden"	FixedRateSwapl/deposit	PASSED
U2	Revert: "Empty swap is not allowed"	FixedRateSwap/swap0To1 FixedRateSwap/swap1To0	FAILED
U3	Revert: "Empty withdrawal is not allowed" "Withdrawal to this is forbidden"	FixedRateSwap/withdraw	PASSED
U4	Revert: "Swap to this is forbidden"	FixedRateSwap/swap0To1For FixedRateSwap/swap1To0For	PASSED

Notes

For failed tests see Appendix A.

5.4 Fuzzy testing

We executed millions of test runs, running for hours, all passed successfully.

Overview

ID	Description	Contract/Function	Result
F1	outputAmount during swaps can not be higher than inputAmount	FixedRateSwapI/swap0To1	PASSED
F2	outputAmount is not 0 on higher inputAmount	FixedRateSwap/swap0To1	PASSED
F3	Possibility of integer overflow/underflow in unchecked blocks	FixedRateSwap/_powerHelper FixedRateSwap/getReturn	PASSED
F4	Contract has same expected behaviour with various decimals in constructor	FixedRateSwap	PASSED

Notes

During testing _powerHelper() function we found one of values:

_powerHelper(3642153733620721l2)

which returns 0 (when calculating manually: 0.250219113). We haven't found it exploitable, just worthy to mention that there can be some "interesting" values causing the return value equal to 0.

6. Conclusion

Wwe had to understand the system and use cases in the first place. The Chief Auditor then started with tool-based analysis, math simulations and manual code review. In parallel, Tester 1 was writing unit tests in Brownie and also running Echidna fuzzer. As a result, we've identified a few problems in 1inch Fixed Rate Swap, which are fairly easy to fix and will make the system more robust.

Appendix A

Demonstration of contract's behaviour leading to a failed U2 test.

```
>>> f.swap0To1(0)
Transaction sent: 0x2efb7e60f9a6562e4e43e88a606c0387296a6efa14d7596f3ea7ef780f04e4c7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 231
  FixedRateSwap.swap0To1 confirmed (Division or modulo by zero)   Block: 295   Gas used:
28250 (0.24%)

<Transaction '0x2efb7e60f9a6562e4e43e88a606c0387296a6efa14d7596f3ea7ef780f04e4c7'>
>>> f.swap0To1(1)
Transaction sent: 0xfe69cd88f880380677f2933f2770889ef9eacf712424405c20f3e741222a12c6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 232
  FixedRateSwap.swap0To1 confirmed (Empty swap is not allowed)   Block: 296   Gas used:
28503 (0.24%)

<Transaction '0xfe69cd88f880380677f2933f2770889ef9eacf712424405c20f3e741222a12c6'>
>>> f.swap0To1(2)
Transaction sent: 0x20d0e11b2b2a92b80efe670bf8ced7fb06a6e3fe06f1c01d16d9b79f24e6d39f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 233
  FixedRateSwap.swap0To1 confirmed   Block: 297   Gas used: 71153 (0.59%)

<Transaction '0x20d0e11b2b2a92b80efe670bf8ced7fb06a6e3fe06f1c01d16d9b79f24e6d39f'>
>>> f.getReturn(USDT,USDC,0)
  File "<console>", line 1, in <module>
  File "brownie/network/contract.py", line 1728, in __call__
    return self.call(*args, block_identifier=block_identifier)
  File "brownie/network/contract.py", line 1532, in call
    raise VirtualMachineError(e) from None
VirtualMachineError: revert
>>> f.getReturn(USDT,USDC,1)
0
>>> f.getReturn(USDT,USDC,2)
1
```

ackee blockchain

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/fc6CRw9n