CYBER UNIT
Cyber Security Strategic Partner
www.cyberunit.tech

Impermax Oracle Smart Contract Audit

Date: January 18, 2020
Report for: Impermax Finance
By: CyberUnit.Tech

This document may contain confidential information about IT systems and the customer's intellectual property and information about potential vulnerabilities and exploitation methods.
The report contains confidential information. This information can be used internally by the customer. The customer can release the information after fixing all vulnerabilities.

Document

Name	Impermax oracle
Platform	EVM
Link	https://github.com/Impermax-Finance/simple-uniswap-oracle
Date	24/12/2020

This document may contain confidential information about IT systems and the intellectual property of the customer and information about potential vulnerabilities and methods of their exploitation.
The report contains confidential information. This information can be used internally by the customer. The customer can release the information after fixing all vulnerabilities.

Document

Name	Impermax-Finance Oracle
Platform	EVM
Link	**https://github.com/Impermax-Finance/simple-uniswap-oracle**
Date	24/12/20

Table of contents

Introduction

This report presents the security assessment findings of the Customer` s smart contract and its code review conducted in December 2020.

Scope

The scope of the project is FinanceOracle smart contract, which can be found by the link below:

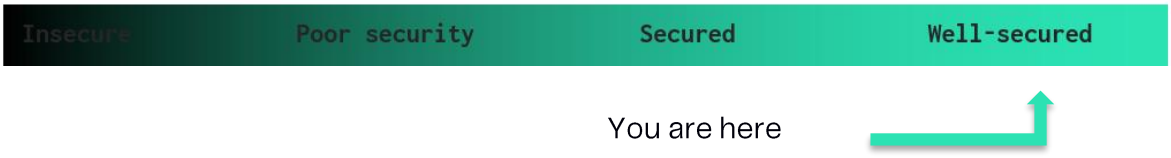
LINK – https://github.com/Impermax-Finance/simple-uniswap-oracle

We have scanned this smart contract for commonly known and more specific vulnerabilities. The following are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited by them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call – Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Customer` s smart contracts are well-secured.



Our team performed an analysis of code functionality, manual audit, and automated checks with Slither and remix IDE (see Appendix B pic 1-2). All issues found during automated analysis were reviewed manually, and application vulnerabilities are presented in the Audit overview section. A general overview is presented in the AS-IS section, and all found issues can be found in the Audit overview section.
We didn't find any low, medium, high, and critical issues in the smart contract.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are essential to fix; however, they can't lead to tokens lose.
Low	Low-level vulnerabilities are mostly related to outdated, unused etc., code snippets that can't have significant impact on execution.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

SimpleUniswapOracle contract consists of the following smart contracts:

1. **UQ112x112**,**IUniswapV2Pair**,**ISimpleUniswapOracle** libraries and interfaces – standard for known as Uniswap v2.
2. **SimpleUniswapOracle** contract – implementation of Uniswap **ISimpleUniswapOracle** smart contract

Contracts from point one

We didn't find any logic differences in contracts from point one than the original Uniswap v2 templates. It's secure.

Contracts from point two
A price oracle based on Uniswap the oracle will always calculate the average price over the last T seconds, where T ≥ MIN_T. In this implementation MIN_T = 3600 seconds. As on Uniswap, anybody can add a new pair to the oracle by passing the address of the UniswapV2 pair to the function initialize().

All data kept inside the oracle smart contract. Uniswap's price0CumulativeLast and the block for each pair timestamp the value saved during the last update and the second-last. It will use these values to calculate the average price in future requests. The current implementation saves final and second-last as A and B, exchanging them at each update. It allows us to reduce the number of write-in storage and minimize gas costs.

After the call of initialize(), the price will not be available for MIN_T seconds, after which the price will always be available. The price for an initialized UniswapV2 pair can be requested by passing the address of that pair to the function getResult(). If T ≥ MIN_T seconds have elapsed since the last update, then the price oracle will return T and price, the average price in the previous T seconds. Otherwise, if not enough time has elapsed since the last update, the oracle will use the second-last update to make the calculations. In the first case, the protocol will also update the pair, while in the second case, the storage will remain unchanged.

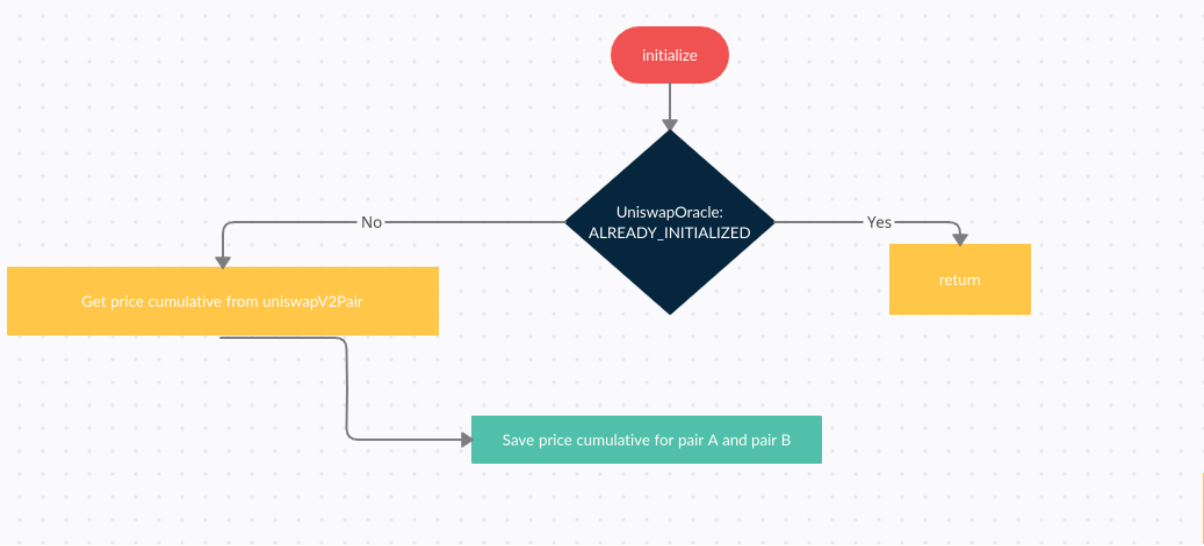
Note: as a convention, it is always calculated the price as token0/token1.

SimpleUniswapOracle contract has several differences compared to the latest Uniswap implementation:

1. **SimpleUniswapOracle** contract inherits ReturnData
2. **toUint224** , **getPriceCumulativeCurrent**,**initialize**,**getResult**
3. **initialize** , **getResult** events added

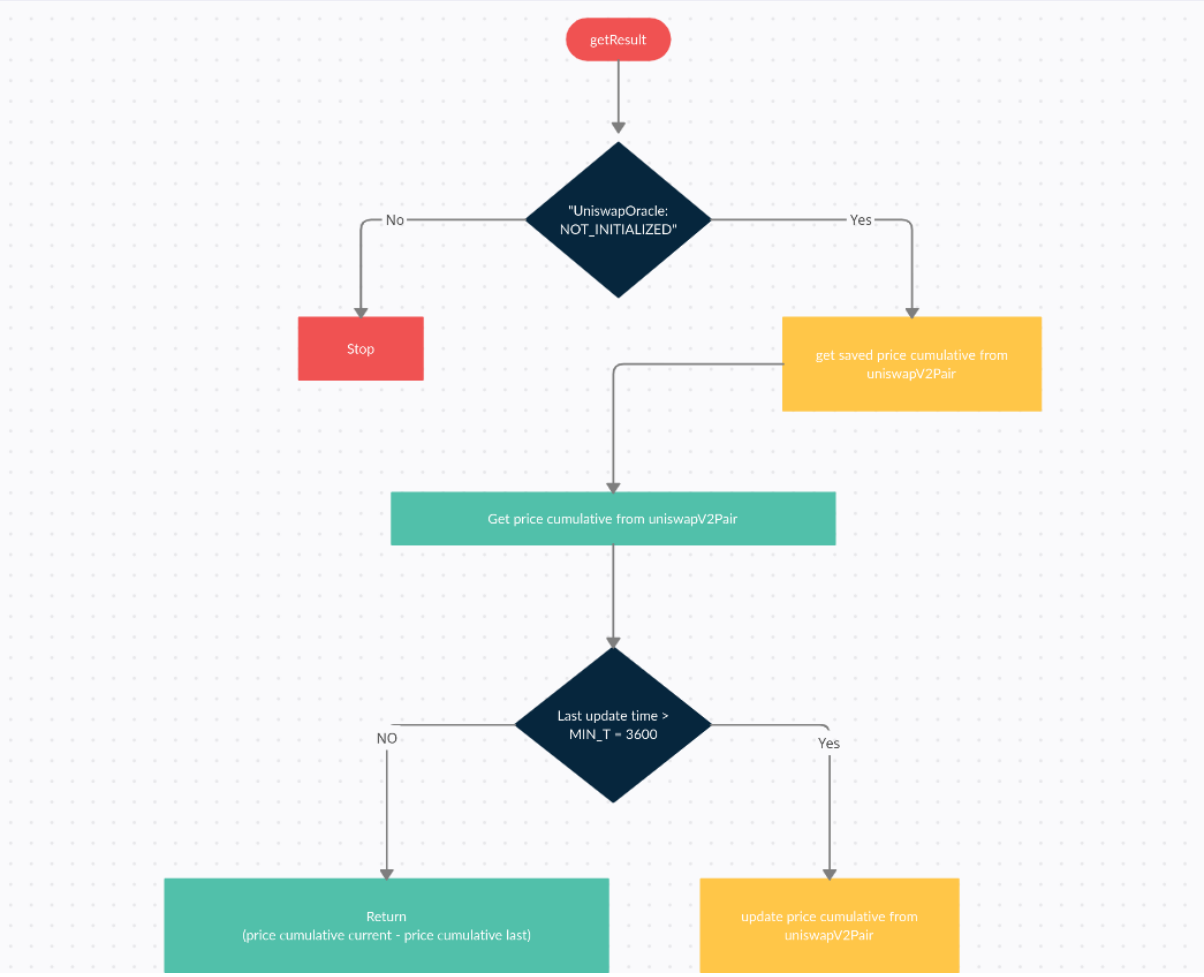
SimpleUniswapOracle contract **initialize** called function with the following parameters:

- uniswapV2Pair (address) **UniswapV2Pair**



getResult function called with the following parameters:

- uniswapV2Pair (address) **UniswapV2Pair**



latestVersion was set at the moment of review.

Note: Contract tests in production are out-of-scope of the current security review.

Audit overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

No low severity vulnerabilities were found.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, a high-level description of functionality was presented in the report's As-is overview section.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The overall quality of the reviewed contracts is good. Security engineers found one low vulnerability, which couldn't have any significant security impact.

Disclaimers

Disclaimer

The smart contracts given for audit had been analyzed following the best industry practices at the date of this report, concerning: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).
The audit makes no statements or warranties on the security of the code. It can also not be considered a sufficient assessment regarding the code's utility and safety, bug-free status, or any other contract statements. While we have done our best to conduct the analysis and produce this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, programming language, and other software related to the smart contract can have their vulnerabilities leading to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix A. Automated tools report

Pic 1. Slither automated report:

```
INFO:Detectors:
SimpleUniswapOracle.toUint224(uint256) (SimpleUniswapOracle.sol#24-27) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(input <= uint224(- 1),UniswapOracle: UINT224_OVERFLOW) (SimpleUniswapOracle.sol#25)
SimpleUniswapOracle.initialize(address) (SimpleUniswapOracle.sol#38-51) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(! pairStorage.initialized,UniswapOracle: ALREADY_INITIALIZED) (SimpleUniswapOracle.sol#40)
SimpleUniswapOracle.getResult(address) (SimpleUniswapOracle.sol#53-86) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(pair.initialized,UniswapOracle: NOT_INITIALIZED) (SimpleUniswapOracle.sol#55)
        - blockTimestamp - updateLast >= MIN_T (SimpleUniswapOracle.sol#63)
        - require(bool,string)(T >= MIN_T,UniswapOracle: NOT_READY) (SimpleUniswapOracle.sol#83)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Function ISimpleUniswapOracle.MIN_T() (interfaces/ISimpleUniswapOracle.sol#5) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:SimpleUniswapOracle.sol analyzed (4 contracts with 46 detectors), 4 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Pic 2. RemixIDE automated report:

Gas requirement of function SimpleUniswapOracle.getResult(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function SimpleUniswapOracle.initialize(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

ISimpleUniswapOracle.getPair(address) : Variables have very similar names priceCumulativeA and priceCumulativeB.

ISimpleUniswapOracle.getPair(address) : Variables have very similar names updateA and updateB.

Use assert(x) if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use require(x) if x can be false, due to e.g. invalid input or a failing external component.
more