CERTIK

Security Assessment

impermax-x-uniswapv2-core

Apr 27th, 2021

Summary

This report has been prepared for impermax-x-uniswapv2-core smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	impermax-x-uniswapv2-core
Description	a DeFi ecosystem that enables liquidity providers to leverage their LP tokens
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/Impermax-Finance/impermax-x-uniswapv2-core
Commits	016eed8f31b8a600446af27da6417ce5d6b0e744

Audit Summary

Delivery Date	Apr 27, 2021
Audit Methodology	Static Analysis
Key Components	

Vulnerability Summary

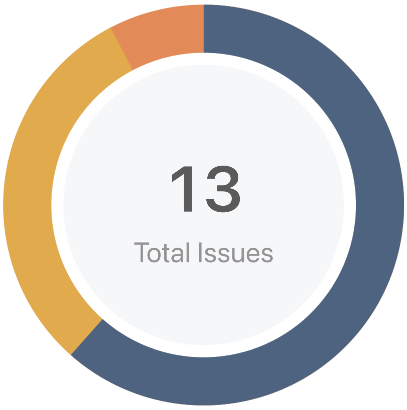
Total Issues	13
● Critical	0
● Major	1
● Minor	4
● Informational	8
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
BAI	BAllowance.sol	6359dd1fd3df9fb6d3b16efd172d38636b55e2570af8fb574d1eeb4d10119a30
BDI	BDeployer.sol	1dda67705e9a4ef1da0bcaef30d10244b098ab2544d42abcb7b77ae2640bc414
BIR	BInterestRateModel.sol	59734f15b721642a0b7634b1a0600f0f83a21da6911c57943c1a3f5776ce4571
BSI	BSetter.sol	61a8f7e7c035a3a055af4618f741c38fe0ea0c28e11cbaca963af89e7abe8252
BSF	BStorage.sol	93e365cd31eea23cb6769137aa757098859bf6fee9cd050d9a5de73a13a777a5
BIF	Borrowable.sol	35ee34b460e4991da9549cef6ad13aa5ccb3cc696de4f50b0d09f2f529eabe48
CDI	CDeployer.sol	e5af9333137e817b14095a6ce7f800591f25e84a1c7e2b8cefa2453761927369
CSI	CSetter.sol	afd226da6aa06f1d8d084cd1767e624b47cf04b0d436bb472ed8b54b06d82cdd
CSF	CStorage.sol	2021d004cc7e518c80e4bd800fcdecf77b7b90e90d57a2905fc8b38be9002d72
CIF	Collateral.sol	05b6dd6ba94fcfe0b5dd22324ce23a46640c7c45b8466d74cf318a0e1fd0f7a0
FIF	Factory.sol	1e89c7e4b59ce40a15935c314f3e1ea982c6f67bfbea96da9edc4498bb75dadb
IER	ImpermaxERC20.sol	67f948f57e61c8fbfc2a269a23841cc9a7bbe36195535015429f31cb09900ec7
PTI	PoolToken.sol	d462d83f01ed88a7753fbb4f9a574bc699293bf12edca4b0768412cfe039fa9b
IBD	interfaces/IBDeployer.sol	7fae6dfe1b583d6986a420ddc4472d8b5a9527d2f5adee9f2d0a176bca182939
IBT	interfaces/IBorrowTracker.sol	93b5a28251422458125850e2a7e16658f4df9d1e780351582fe127b3dd71dc8a
IBI	interfaces/IBorrowable.sol	55f45e88a722c94d5a1f46ceb1b67ce26f322ed7628d916aec9450f3a21ca2bb
ICD	interfaces/ICDeployer.sol	ed4fd37b6e86b4e74ccb4015d832b54bc0e1c5f85e4195dda34c025b98527fd8
ICI	interfaces/ICollateral.sol	254960601c590da8adfa086b059399a9bed1b29e57ffc40663505fd9e6e3074a
IEC	interfaces/IERC20.sol	db8b0c6761ec7c98d8011e4ff0e12df131e19e737066017696f60821ef3a1ff3
IFI	interfaces/IFactory.sol	908ba60ced8c7b82d475a4ee0563e788aeb10841d91e2a172e29d8245f8a0ce3
IIC	interfaces/IImpermaxCallee.sol	d6218bcf6d1a8098a4756fd04c7140fc21c22ce0f47e5a9c4b6e059dc200fd19
IPT	interfaces/IPoolToken.sol	7a4e0d680f21ad528de23f155f4ff490300960deffd702ee13977bde86ca5002

ID	file	SHA256 Checksum
ISU	interfaces/ISimpleUniswapOracle.sol	8e03e24c56464f4ca5a9c567c6bed17db8d6fece9c999378dad1673c89be8fa9
IUV	interfaces/IUniswapV2Factory.sol	18abdd0001a7550378dde2770dbd4f0857e46ca0406f206b4dd211a93c7071a3
IUP	interfaces/IUniswapV2Pair.sol	5d6ac91d7afdabff1a103d1070faf1e12a82d9f283484f0aa23281484e01f5fd
MIF	libraries/Math.sol	b92f613c3eb2e629af384e5b9016ee2f23cdcdcfe3e48c779a70ab675053ede9
SMI	libraries/SafeMath.sol	f8549f138fc64c7c6f43beac66e7045833a2d7df125c2f41637cc5dda66f7790
UQI	libraries/UQ112x112.sol	506750960e850aff49f278a51a8135ca4bd2de555cb23eef3f45d9917fa71284

Findings



🟥 Critical	**0** (0.00%)
🟧 Major	**1** (7.69%)
🟨 Minor	**4** (30.77%)
🟦 Informational	**8** (61.54%)
🟩 Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
BDI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
BIF-01	Unknown Implementation of `trackBorrow` Function	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
BIF-02	Unknown Implementation of `balanceOf` Function	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
BIF-03	Lack of Input Validation	Logical Issue	● Informational	ⓘ Acknowledged
BIF-04	Lack of Input Validation	Logical Issue	● Informational	ⓘ Acknowledged
BIR-01	Division Before Multiplication	Mathematical Operations	● Informational	ⓘ Acknowledged
BSI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
BSI-02	Unknown Implementation of `trackBorrow` Function	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
BSI-03	Unknown Implementation of `balanceOf` Function	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**

ID	Title	Category	Severity	Status
CDI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
CSI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
FIF-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
PTI-01	Unknown Implementation of `balanceOf` Function	**Centralization / Privilege**	● **Major**	ⓘ **Acknowledged**

BDI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	BDeployer.sol: 14	ⓘ Acknowledged

Description

Missing validation for the input variables `uniswapV2Pair` in function `deployCollateral()`

Recommendation

We advise the client to ensure these input variables are not equal to `address(0)`

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

BIF-01 | Unknown Implementation of `trackBorrow` Function

Category	Severity	Location	Status
Centralization / Privilege	● **Minor**	**Borrowable.sol: 69**	ⓘ **Acknowledged**

Description

`_setBorrowTracker()` function can set `borrowTracker` to any contract address that is implemented from `IBorrowTracker` interface by owner. As result, the invocation of `IBorrowTracker(_borrowTracker).trackBorrow()` in function `_trackBorrow()` may bring dangerous effects as it is unknown to the user.

Recommendation

We advise the client to check and ensure the contract at address `borrowTracker` is a standard BorrowTracker smart contract that follows the `IBorrowTracker` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Impermax]`: Took note of the recommendation.

BIF-02 | Unknown Implementation of `balanceOf` Function

Category	Severity	Location	Status
Centralization / Privilege	● Minor	Borrowable.sol: 112, 129	ⓘ Acknowledged

Description

`_initialize()` function can set `underlying` to any contract address that is implemented from `IERC20` interface by owner. As result, the invocation of `IERC20(underlying).balanceOf()` in function `borrow()` and in `liquidate()` function may bring dangerous effects as it is unknown to the user.

Recommendation

We advise the client to check and ensure the contract at address `underlying` is a standard ERC20 smart contract that follows the `IERC20` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Impermax]`: Impermax is a permissionless protocol. Similarly to Uniswap, a hacker could use Impermax to create pairs with malicious ERC20 tokens. This is a known issue which we cover in the risk section of our app: https://app.impermax.finance/risks

BIF-03 | Lack of Input Validation

Category	Severity	Location	Status
Logical Issue	● Informational	Borrowable.sol: 104	Acknowledged

Description

The input value of `borrower` and `receiver` should not be same address

Recommendation

We advise the client to adopt a input validator to prevent any inputs which `borrower` and `receiver` are the same.

```
require(borrower != receiver, "borrower and receiver are the same");
```

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

BIF-04 | Lack of Input Validation

Category	Severity	Location	Status
Logical Issue	● Informational	Borrowable.sol: 128	ⓘ Acknowledged

Description

The input value of `borrower` and `receiver` should not be same address

Recommendation

We advise the client to adopt a input validator to prevent any inputs which `borrower` and `liquidator` are the same.

```
require(borrower != liquidator, "borrower and liquidator are the same");
```

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

BIR-01 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	● Informational	BInterestRateModel.sol: 33, 37	ⓘ Acknowledged

Description

Mathematical operations in the aforementioned lines perform divisions before multiplications. Performing multiplication before division can sometimes avoid loss of precision.

Recommendation

We recommend applying multiplications before divisions to avoid loss of precision.

Alleviation

`[Impermax]`: Division before multiplication: in the cases where we've done this the division was always preceded by another multiplication. So we're doing multiplication, then division, then multiplication. If we did multiplication, then multiplication, then division there would be the possibility to overflow.

BSI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	BSetter.sol: 29~30, 54	ⓘ Acknowledged

Description

Missing validation for the input variables `_underlying`, `_collateral` in constructor of `BSetter` contract and input variable `newBorrowTracker` in function `_setBorrowTracker()`

Recommendation

We advise the client to ensure these input variables are not equal to `address(0)`

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

BSI-02 | Unknown Implementation of `trackBorrow` Function

Category	Severity	Location	Status
Centralization / Privilege	● Minor	BSetter.sol: 52	ⓘ Acknowledged

Description

`_setBorrowTracker()` function can set `borrowTracker` to any contract address that is implemented from `IBorrowTracker` interface by owner. As result, the invocation of `IBorrowTracker(_borrowTracker).trackBorrow()` in function `_trackBorrow()` may bring dangerous effects as it is unknown to the user.

Recommendation

We advise the client to check and ensure the contract at address `borrowTracker` is a standard BorrowTracker smart contract that follows the `IBorrowTracker` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Impermax]`: Took note of the recommendation.

BSI-03 | Unknown Implementation of `balanceOf` Function

Category	Severity	Location	Status
Centralization / Privilege	● Minor	BSetter.sol: 21	ⓘ Acknowledged

Description

`_initialize()` function can set `underlying` to any contract address that is implemented from `IERC20` interface by owner. As result, the invocation of `IERC20(underlying).balanceOf()` in function `borrow()` and in `liquidate()` function may bring dangerous effects as it is unknown to the user.

Recommendation

We advise the client to check and ensure the contract at address `underlying` is a standard ERC20 smart contract that follows the `IERC20` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Impermax]`: Impermax is a permissionless protocol. Similarly to Uniswap, a hacker could use Impermax to create pairs with malicious ERC20 tokens. This is a known issue which we cover in the risk section of our app: https://app.impermax.finance/risks

CDI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	CDeployer.sol: 14	ⓘ Acknowledged

Description

Missing validation for the input variables `uniswapV2Pair` in function `deployCollateral()`

Recommendation

We advise the client to ensure these input variables are not equal to `address(0)`

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

CSI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	CSetter.sol: 28~30	ⓘ Acknowledged

Description

Missing validation for the input variables `_underlying`, `_borrowable0`,`_borrowable1` in constructor of `CSetter` contract.

Recommendation

We advise the client to ensure these input variables are not equal to `address(0)`

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

FIF-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	Factory.sol: 46~52	ⓘ Acknowledged

Description

Missing validation for the input variables in constructor of `Factory` contract.

Recommendation

We advise the client to ensure these input variables are not equal to `address(0)`

Alleviation

`[Impermax]`: Our design choice for the core contracts was to not do input validation for core contracts unless they were absolutely required for security reasons. Instead we have decided to keep the contracts as simple and flexible as possible. Core contracts must be called by an external contract which can then implement input validation.

PTI-01 | Unknown Implementation of `balanceOf` Function

Category	Severity	Location	Status
Centralization / Privilege	● Major	PoolToken.sol: 83, 44, 72, 30	ⓘ Acknowledged

Description

Due to the unknown address of `underlying`, the invocation of `IERC20(underlying).balanceOf()` in function `update()`, `mint()` and `skim()`, and `underlying.call(abi.encodeWithSelector())` in `_safeTransfer()` function may bring dangerous effects as it is unknown to the user.

Recommendation

We advise the client to check and ensure the contract at address `underlying` is a standard ERC20 smart contract that follows the `IERC20` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Impermax]`: Impermax is a permissionless protocol. Similarly to Uniswap, a hacker could use Impermax to create pairs with malicious ERC20 tokens. This is a known issue which we cover in the risk section of our app: https://app.impermax.finance/risks

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.