CERTIK

Security Assessment

IMX

Apr 27th, 2021

Summary

This report has been prepared for IMX 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	IMX
Description	a DeFi ecosystem that enables liquidity providers to leverage their LP tokens
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/Impermax-Finance/IMX
Commits	1. 6c8b87761c178a0755603cfe867c2ba395ffe25c 2. dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01

Audit Summary

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

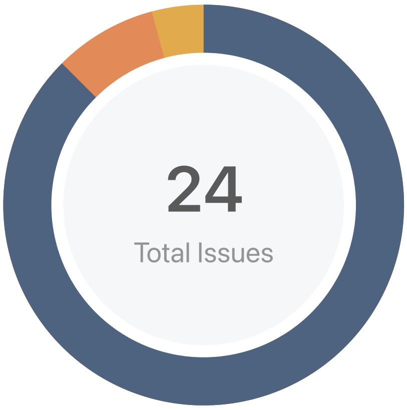
Vulnerability Summary

Total Issues	24
● Critical	0
● Major	2
● Minor	1
● Informational	21
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
CAI	ClaimAggregator.sol	94bf022aa56ff0daf7f76f2d3adc1f0aa966c527e4bbdf6bb049714238e3f36b
DIM	Distributor.sol	22d8374e4ab3ea5e20fb7ea3ba768e94e5101c843eeefaaae7e1f948b6055e20
FPI	FarmingPool.sol	fe6a374356ec6c00b65bd06e8a730d333d0af90179eb8ab5dfcf23d7e1b7ede8
IIM	Imx.sol	f2661fcd451eed5800195d928c084f712ad91de972109917dc73079e95904676
IDI	InitializedDistributor.sol	61adb96e7b13540cd10c9931cf5f9927e907fec7f7dda88ee919d5c0b1cf6c03
ODI	OwnedDistributor.sol	d1352a2603e74d1cf8207b6f74543ac936a9412212560d1c8f09fbe475a94a18
VIM	Vester.sol	b76fcca3a2d687c1b9e60a550a884870fbb3b7196ed33ee247cca66f1c46959f
VSI	VesterSale.sol	a2aee6f3870d86ac604462fe0ee7fce5cfebef04d469a01aeaa9e2b03a0ba650
VSM	VesterStepped.sol	30e0fa66a9db51f5363a114d7eccc9d7e03af928faca34cde02c1b13d0ea79c6
IBT	interfaces/IBorrowTracker.sol	473c186694527cd445ae74865869a7dc667edd981ac2f2f97b9db8fdacece62f
IBI	interfaces/IBorrowable.sol	5e27dad67a521f96937f916d1e645edf57b4f0c06f71b01fa9bea5c753ff9bd8
ICI	interfaces/IClaimable.sol	18d4bd972e9b9c97c25852fadbbfe307684463630fa6651eb78c43ed8d0a4578
IFP	interfaces/IFarmingPool.sol	4702107e901d90cdc40cd72ff8bc3631dd017734084a11075bd7ab4508657a47
III	interfaces/IImx.sol	184685aaa70cca3768efe7e8e4789892fca301e942ded3f633a58759fe49e1a8
IVI	interfaces/IVester.sol	618786b136bab758494a0856e7062a4f43ec6b852fb8cdf5d62ac88fbdaac43a
MIM	libraries/Math.sol	628baf8cd54ce809dfdd4bc2d73a1267c481af79f014afd22775cc6f650b6872
SMI	libraries/SafeMath.sol	6e0e7c72ac7a0338f4200a77e00135d9c7ee97975254417a1a0c3ed1028edf69

Findings



🟥 Critical	**0** (0.00%)
🟧 Major	**2** (8.33%)
🟨 Minor	**1** (4.17%)
🟦 Informational	**21** (87.50%)
🟩 Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
CAI-01	Proper Usage of `public` and `external` type	Gas Optimization	● Informational	✓ Resolved
CAI-02	SafeMath Not Used	Language Specific	● Informational	ⓘ Acknowledged
DIM-01	Unknown Implementation of `claim` Function	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
DIM-02	Return Value Not Checked	Logical Issue	● Informational	ⓘ Acknowledged
DIM-03	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
DIM-04	Proper Usage of `public` and `external` type	Gas Optimization	● Informational	✓ Resolved
DIM-05	SafeMath Not Used	Language Specific	● Informational	ⓘ Acknowledged
FPI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
FPI-02	Proper Usage of `public` and `external` type	Gas Optimization	● Informational	✓ Resolved
FPI-03	SafeMath Not Used	Language Specific	● Informational	✓ Resolved
FPI-04	SafeMath Not Used	Language Specific	● Informational	✓ Resolved
IDI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
IDI-02	SafeMath Not Used	Language Specific	● Informational	✓ Resolved
ODI-01	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged

ID	Title	Category	Severity	Status
ODI-02	Centralized Risk	Control Flow	● Major	ⓘ Acknowledged
ODI-03	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
VIM-01	Return Value Not Checked	Logical Issue	● Informational	ⓘ Acknowledged
VIM-02	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
VIM-03	Lack of Input Validation	Volatile Code	● Informational	ⓘ Acknowledged
VIM-04	Centralized Risk	Control Flow	● Major	ⓘ Acknowledged
VIM-05	SafeMath Not Used	Language Specific	● Informational	ⓘ Acknowledged
VIM-06	SafeMath Not Used	Language Specific	● Informational	✓ Resolved
VSI-01	SafeMath Not Used	Language Specific	● Informational	✓ Resolved
VSM-01	SafeMath Not Used	Language Specific	● Informational	ⓘ Acknowledged

CAI-01 | Proper Usage of `public` and `external` type

Category	Severity	Location	Status
Gas Optimization	● Informational	ClaimAggregator.sol: 11, 17	✓ Resolved

Description

Public functions that are never called by the contract could be declared external . When the inputs are arrays external functions are more efficient than “public” functions.

Recommendation

Consider using the external attribute for functions never called from the contract.

Alleviation

`[Impermax]`: Solved in commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

CAI-02 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	ClaimAggregator.sol: 19	ⓘ Acknowledged

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: The sum operator is adding amounts of tokens. The sum will never overflow since there will never be more than 2**256 tokens in circulation.

DIM-01 | Unknown Implementation of `claim` Function

Category	Severity	Location	Status
Centralization / Privilege	● Minor	Distributor.sol: 38	ⓘ Acknowledged

Description

As a parameter, `claimable` can be any contract address that is implemented from the `IClaimable` interface. As a result, the invocation of `IClaimable(claimable).claim()` in function `updateShareIndex()` may bring dangerous effects as it is unknown to the user.

Recommendation

We advise the client to check and ensure the contract at address `claimable` is a standard smart contract that follows the `IClaimable` interface with correct logic implementation as designed by the client.

Alleviation

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

DIM-02 | Return Value Not Checked

Category	Severity	Location	Status
Logical Issue	● Informational	Distributor.sol: 59	ⓘ Acknowledged

Description

The return value of invocation 'IImx(imx).transfer()' is not checked.

Recommendation

We advise the client to check the return value of 'IImx(imx).transfer()' to make sure the invocation is successful.

Alleviation

`[Impermax]`: In our IMX contract the function transfer() always returns true when it doesn’t revert.

DIM-03 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	Distributor.sol: 28	ⓘ Acknowledged

Description

Missing validation for the input variables `imx_`, `claimable_` in constructor of `Distributor` 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.

DIM-04 | Proper Usage of `public` and `external` type

Category	Severity	Location	Status
Gas Optimization	● Informational	Distributor.sol: 64	✓ Resolved

Description

Public functions that are never called by the contract could be declared external . When the inputs are arrays external functions are more efficient than “public” functions.

Recommendation

Consider using the external attribute for functions never called from the contract.

Alleviation

`[Impermax]`: Solved in commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

DIM-05 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	Distributor.sol: 49	ⓘ Acknowledged

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: The sum operator is adding amounts of tokens. The sum will never overflow since there will never be more than 2**256 tokens in circulation.

FPI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	FarmingPool.sol: 26	ⓘ Acknowledged

Description

Missing validation for the input variables `imx_`, `claimable_`,`borrowable_`,`vester_` in constructor of `FarmingPool` 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.

FPI-02 | Proper Usage of `public` and `external` type

Category	Severity	Location	Status
Gas Optimization	● Informational	FarmingPool.sol: 70, 75, 80	✓ Resolved

Description

Public functions that are never called by the contract could be declared external . When the inputs are arrays external functions are more efficient than “public” functions.

Recommendation

Consider using the external attribute for functions never called from the contract.

Alleviation

`[Impermax]`: Solved in the commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

FPI-03 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	FarmingPool.sol: 35	✓ Resolved

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Solve in the commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

FPI-04 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	FarmingPool.sol: 64	✓ Resolved

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Solve in the commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

IDI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	InitializedDistributor.sol: 13	ⓘ Acknowledged

Description

Missing validation for the input variables `imx_`, `claimable_`, in constructor of `InitializedDistributor` 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.

IDI-02 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	InitializedDistributor.sol: 22	✓ Resolved

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Solve in the commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

ODI-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	OwnedDistributor.sol: 11	ⓘ Acknowledged

Description

Missing validation for the input variables `imx_`, `claimable_`,`admin_` in constructor of `OwnedDistributor` 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.

ODI-02 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Major	OwnedDistributor.sol: 24, 19	ⓘ Acknowledged

Description

`admin` is an important role in the contract. The `admin` address can modify the value of `shares` of a specific `account` by calling the function `editRecipient()`

Recommendation

We advise the client to carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt Timelock with reason delay to allow the user to withdraw their funds, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Alleviation

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

ODI-03 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	OwnedDistributor.sol: 26	ⓘ Acknowledged

Description

Missing validation for the input variable `admin_` in function `setAdmin()`.

Recommendation

We advise the client to ensure this input variable is 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.

VIM-01 | Return Value Not Checked

Category	Severity	Location	Status
Logical Issue	● Informational	Vester.sol: 67	ⓘ Acknowledged

Description

The return value of invocation 'IImx(imx).transfer()' is not checked.

Recommendation

We advise the client to check the return value of 'IImx(imx).transfer()' to make sure the invocation is successful.

Alleviation

`[Impermax]`:In our IMX contract the function transfer() always returns true when it doesn’t revert.

VIM-02 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	Vester.sol: 23	ⓘ Acknowledged

Description

Missing validation for the input variables `imx_`, `recipient_` in constructor of `Distributor` 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.

VIM-03 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	Vester.sol: 70	ⓘ Acknowledged

Description

Missing validation for the input variable `recipient_` in function `setRecipient()`

Recommendation

We advise the client to ensure this input variable is 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.

VIM-04 | Centralized Risk

Category	Severity	Location	Status
Control Flow	● Major	Vester.sol: 59, 63, 67, 70	ⓘ Acknowledged

Description

`recipient` is an important role in the contract. The `recipient` address can transfer all amount of tokens to address `recipient` by calling the function `claim()`

Recommendation

We advise the client to carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt Timelock with reason delay to allow the user to withdraw their funds, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Alleviation

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

VIM-05 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	Vester.sol: 45, 46, 47	ⓘ Acknowledged

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Operations never overflow. We don’t use SafeMath to optimize gas costs.

VIM-06 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	Vester.sol: 54	✓ Resolved

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Solve in the commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

VSI-01 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	VesterSale.sol: 18	✓ Resolved

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Solve in the commit `dec54d7c4b1f4277cb451c01b5aaf97db2a2fd01`

VSM-01 | SafeMath Not Used

Category	Severity	Location	Status
Language Specific	● Informational	VesterStepped.sol: 19~20	ⓘ Acknowledged

Description

SafeMath from OpenZeppelin is not used in the following functions making them possible for overflow, which will lead to incorrect results

Recommendation

We advise the client to adopt OpenZeppelin's SafeMath library for all of the `uint` operations. Considering use OpenZeppelin's SafeMath library for all of the `uint` operations throughout the contract.

Alleviation

`[Impermax]`: Operations never overflow. We don’t use SafeMath to optimize the gas cost

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.