CERTIK

Security Assessment

MulanSwap

Apr 13th, 2021

Summary

This report has been prepared for MulanSwap 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	MulanSwap
Description	A decentralised exchange that utilizes an automated liquidity protocol powered (AMM) that uses the constant-product invariant and implemented in a system of non-upgradeable smart contracts on the Ethereum blockchain.
Platform	Ethereum
Language	Solidity
Codebase	1. https://etherscan.io/address/0xba1498C77cfba69Dd58362d256c593140B7e7164#code 2. https://etherscan.io/address/0x8F589dcef8bC9564C17FBa05ab6204e9793C9223#code 3. https://etherscan.io/address/0x7dFB72A2AAd08C937706f21421B15bFC34Cba9ca#code
Commits	mulan

Audit Summary

Delivery Date	Apr 13, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

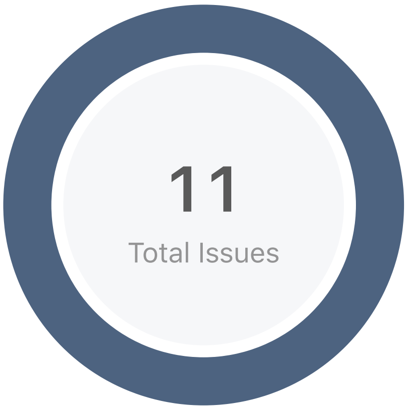
Vulnerability Summary

Total Issues	11
● Critical	0
● Major	0
● Minor	0
● Informational	11
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
BTE	ERC20/BasicToken.sol	dde32d0bba9da74c52e0710e431996308e472f2b07d468c69609e17a3c082e0a
CTE	ERC20/CapToken.sol	b51140b0720ddecee3823ee6a523875894f75d05fdbcc22082ddadf06f3c11f3
CTR	ERC20/CappedToken.sol	c937963bb37204634a4380d6a508728463a219f0c5334e20c765602ecbab4846
ERC	ERC20/ERC20.sol	1570b37daa43d61c3f045639f96f63ca687ac0a8444ff944cd1ed1c33c0141e9
ERB	ERC20/ERC20Basic.sol	8f09e53364787fdff9a9f701c4c5c35b8786d26aed5cce84ca460cd854e8a130
MTE	ERC20/MintableToken.sol	17bd054d51a9da8a42e807a30e4da006b3ce6b57212b773f1001ac0c9abb707d
OER	ERC20/Ownable.sol	35feff96ea2ff782dfa0d35815b2d394cff31bbb2270b77aab4abac1bf6e4b9b
SME	ERC20/SafeMath.sol	2992be99ec79983fab97b08158bbad475f55e02ec8c5293d663fa124a9b75c66
STE	ERC20/StandardToken.sol	17f3420015158148d711851a1f8a266ca417d25645f8ad37569ed9ba34ad351b
MVF	MulanV2Factory.sol	dcb6b1eaac6635fecfb59aee663e3dbeee06bf7294fe3b56220e46345c206de4
MVR	MulanV2Router02.sol	ccdea3d52eda8be86402e2fccf0e29913c33e5e5992ba37efe2c7954c395ab37

Findings



🟥	Critical	**0** (0.00%)
🟧	Major	**0** (0.00%)
🟨	Minor	**0** (0.00%)
🟦	Informational	**11** (100.00%)
🟩	Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
BTE-1	Proper Usage of "public" and "external"	Optimization	● Informational	ⓘ Acknowledged
CTE-1	State variables that could be declared constant	Gas Optimization	● Informational	ⓘ Acknowledged
MVF-1	Incorrect Naming Convention Utilization	Coding Style	● Informational	ⓘ Acknowledged
MVF-2	Weak PRNG	Optimization	● Informational	ⓘ Acknowledged
MVF-3	Missing Emit Events	Optimization	● Informational	ⓘ Acknowledged
MVF-4	Check Zero Address	Optimization	● Informational	ⓘ Acknowledged
MVR-1	Incorrect Naming Convention Utilization	Coding Style	● Informational	ⓘ Acknowledged
MVR-2	Proper Usage of "public" and "external"	Optimization	● Informational	ⓘ Acknowledged
MVR-3	Uninitialized local variables	Coding Style	● Informational	ⓘ Acknowledged
STE-1	Incorrect Naming Convention Utilization	Coding Style	● Informational	ⓘ Acknowledged
STE-2	Proper Usage of "public" and "external"	Optimization	● Informational	ⓘ Acknowledged

BTE-1 | Proper Usage of "public" and "external"

Category	Severity	Location	Status
Optimization	● Informational	ERC20/BasicToken.sol: 22~24, 46~48	ⓘ Acknowledged

Description

The `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.

Examples: Functions like : `quote()`, `getAmountOut()`, and `getAmountIn()`

Recommendation

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

Alleviation

No alleviation.

CTE-1 | State variables that could be declared constant

Category	Severity	Location	Status
Gas Optimization	● Informational	ERC20/CapToken.sol: 8~10	ⓘ Acknowledged

Description

Constant state variables should be declared constant to save gas.

Recommendation

Add the constant attributes to state variables that never change.

Alleviation

No alleviation.

MVF-1 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	● Informational	MulanV2Factory.sol: 38, 39, 56, 126, 621	ⓘ Acknowledged

Description

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to:

https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions

Functions other than constructors should use mixedCase. Examples: Functions like: `DOMAIN_SEPARATOR()`, `PERMIT_TYPEHASH()`, `MINIMUM_LIQUIDITY()`, `WETH()`

Parameter should use mixedCase.

Examples: Parameter like: `_token0`, `_token1`, `WETH`

Inside each contract, library or interface, use the following order: Type declarations, State variables, Events, Functions.

Events definition should be in front of function definitions.

Recommendation

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

No alleviation.

MVF-2 | Weak PRNG

Category	Severity	Location	Status
Optimization	● Informational	MulanV2Factory.sol: 268~281, 191~203	ⓘ Acknowledged

Description

Weak PRNG due to a modulo on block.timestamp, now or blockhash. These can be influenced by miners to some extent so they should be avoided.

Recommendation

Do not use block.timestamp, now or blockhash as a source of randomness.

Alleviation

No alleviation.

MVF-3 | Missing Emit Events

Category	Severity	Location	Status
Optimization	● Informational	MulanV2Factory.sol: 432, 437	ⓘ Acknowledged

Description

Several sensitive actions are defined without event declarations.

Examples:

Functions like `setFeeTo()`, `setFeeToSetter()`.

Recommendation

Consider adding events for sensitive actions, and emit it in the function.

Alleviation

No alleviation.

MVF-4 | Check Zero Address

Category	Severity	Location	Status
Optimization	● Informational	MulanV2Factory.sol: 261~265, 432, 437	ⓘ Acknowledged

Description

The parameter of address should be checked for zero address in function.

Example:

```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'MulanV2: FORBIDDEN');
        feeTo = _feeTo;
    }
```

Recommendation

We recommend to add below checks.

```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'MulanV2: FORBIDDEN');
        require(_feeTo != address(0), "Address zero is forbbiden");
        feeTo = _feeTo;
    }
```

Alleviation

No alleviation.

MVR-1 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	● Informational	MulanV2Router02.sol: 38, 39, 235	ⓘ Acknowledged

Description

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to:

https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions

Functions other than constructors should use mixedCase. Examples: Functions like: `DOMAIN_SEPARATOR()`, `PERMIT_TYPEHASH()`, `MINIMUM_LIQUIDITY()`, `WETH()`

Parameter should use mixedCase.

Examples: Parameter like: `_token0`, `_token1`, `WETH`

Inside each contract, library or interface, use the following order: Type declarations, State variables, Events, Functions.

Events definition should be in front of function definitions.

Recommendation

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

No alleviation.

MVR-2 | Proper Usage of "public" and "external"

Category	Severity	Location	Status
Optimization	● Informational	MulanV2Router02.sol: 622~624, 626~634, 636~644, 646 ~654, 656~664	ⓘ Acknowledged

Description

The `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.

Examples: Functions like : `quote()`, `getAmountOut()`, and `getAmountIn()`

Recommendation

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

Alleviation

No alleviation.

MVR-3 | Uninitialized local variables

Category	Severity	Location	Status
Coding Style	● Informational	MulanV2Router02.sol: 432, 541, 742	ⓘ Acknowledged

Description

Uninitialized local variables.

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Alleviation

specify

STE-1 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	● Informational	ERC20/StandardToken.sol: 26~28, 53	ⓘ Acknowledged

Description

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to:

https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions

Functions other than constructors should use mixedCase. Examples: Functions like: `DOMAIN_SEPARATOR()`, `PERMIT_TYPEHASH()`, `MINIMUM_LIQUIDITY()`, `WETH()`

Parameter should use mixedCase.

Examples: Parameter like: `_token0`, `_token1`, `WETH`

Inside each contract, library or interface, use the following order: Type declarations, State variables, Events, Functions.

Events definition should be in front of function definitions.

Recommendation

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

No alleviation.

STE-2 | Proper Usage of "public" and "external"

Category	Severity	Location	Status
Optimization	● Informational	ERC20/StandardToken.sol: 53~57, 107~122	ⓘ Acknowledged

Description

The `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.

Examples: Functions like : `quote()`, `getAmountOut()`, and `getAmountIn()`

Recommendation

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

Alleviation

No alleviation.

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.