CERTIK

Security Assessment

OccamX

Apr 15th, 2022

Table of Contents

Summary

This report has been prepared for OccamX to discover issues and vulnerabilities in the source code of the OccamX project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing 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;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	OccamX
Platform	EVM Compatible
Language	Solidity
Codebase	https://github.com/OccamX-MilkomedaDEX/Milkomeda-OCCAMX-sc/tree/audit_7_march
Commit	233d7a724f438baf2297d9b6406c6944c5882817

Audit Summary

Delivery Date	Apr 15, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	2	0	0	2	0	0	0
Medium	1	0	0	0	0	0	1
Minor	2	0	0	2	0	0	0
Informational	8	0	0	6	0	0	2
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IPM	interfaces/IPair.sol	c702c9100a2d334d71962a0797ca53dd35406cf1bd04c9dc0f8490b2fbf3ce72
IWA	interfaces/IWADA.sol	81a69703e6dd0910035b729ae40b61b5f335cdcfeebdaffa56e1ccc15fd2d412
MOX	libraries	
IER	interfaces/IERC20.sol	d79228f24fab9904e1b0d29ec0d5d0200268608861596acecd4441b95d0e13b2
OMO	Ownable.sol	9fb68dc3d03ba79c6286e5f14db100881e99fffa35bb9b9951d7d80fc8d5b8d3
IEM	interfaces/IERC2612.sol	f4730ad5c18d00a14f01433e99d7e22c4f7e66b33583568e3da206b0d66e6396
MOA		
ICM	interfaces/ICallee.sol	cbb430c097d252981c81710098a826edc82093b513c46b950f6a5c42d0c27a09
PTM	ProtocolToken.sol	9f243f7a0f197256d592b051d240dc9a1035fc925d2def60cfd8fe5231b5870c
IEV	interfaces/V1/IExchange.sol	d5a9d1a55121e5ad22da525ce6b61776508ba47f2e07df58a0d5333ad386182d
MOM	interfaces	
THM	libraries/TransferHelper.sol	764551cf18a6fe37504c92874c20a0299cc7e10b0c23cbfefe2293b430dc4533
IEP	interfaces/IERC20Permit.sol	09993d4133a79688e1aa00b4183d3cb33706e10c034572199249a27d340a8775

ID	File	SHA256 Checksum
CMO	Collector.sol	1651dace9fedf255a14162d9aa05812eb2f52d7cea35e2224cd8f97a72b20ce9
VMO	interfaces/V1	
IEC	interfaces/IERC20Burnable.sol	1cea888edc14b715ce2c0cff09f6051d4bfe5ba727016a3ddbec92cde52773b1
ERC	ERC20Permit.sol	0888e6532a06b37e7bdfd8520621dec13f2a3793533e60fa003c7fab22d72593
CHM	CalHash.sol	f0aa0ff2cde564f7ce101010b37286fd9c6215cd457b13b1498a6d4ec87ab917
RMO	Router01.sol	177d1460b0b1213db9c488bd040c5df9466c4268e1975598b8c87167e7d4e1ad
IWD	interfaces/IWADA10.sol	a6c9cca4751bc53ac0b37d4f6ad64fba2df72c47a679783f2bdcbbd5bf79a737
PMO	Pair.sol	fbd646322d6afdc1b9deb3372671cf91ec518a7334e7754f0ebacf1713ca7a6f
MOC	$/github/CertiKProject/certik-audit-projects/08e1acb77bede3fd414f57e6587fb26320adb1ba/projects/Milkomeda-OCCAMX-sc-233d7a724f438baf2297d9b6406c6944c5882817	
UQM	libraries/UQ112x112.sol	821ab924e12321a28d349108bb65fbf8956c6194d15ead3a261f58d73a919003
IFM	interfaces/IFactory.sol	91a1af70629ce8b3bcf4403fbc14548ced717ded4278065a2c18fcab2ff0206b
IRO	interfaces/IRouter02.sol	420c03d3c921faf2596501f1141c33a23fa2455e7f279163c318ab5c987904bf
SER	libraries/SafeERC20.sol	793b1c2f01399b5b2d8540c60b3f3775f2dfd15a33406cbef035c5d750b1c9c1

ID	File	SHA256 Checksum
RMC	Router02.sol	e79513b4d2889fa99cf79867a2983fccb93fa4671e4af2341185192f1e089a14
CKP	$/github/CertiKProject/certik-audit-projects/08e1acb77bede3fd414f57e6587fb26320adb1ba/projects	
OLM	libraries/OracleLibrary.sol	3e60fe579eaf2aaba1387069e9c56f674c0bcafe43f876d11b49baea7c394493
MMO	libraries/Math.sol	8be17db8484803ce35e9277153f2d58d9b8757a4f84ed2a054e7f9264b442283
FMO	Factory.sol	87475e5ea7ff8358ed4cf881418ac2de4a5464043a9cf16c8f52f8bb8f420e81
IRM	interfaces/IRouter01.sol	7036fc8625dcdb104fc6144ac06245a0f5ed8a62338552151474a1974d6e8ed2
LMO	libraries/Library.sol	e0a0c476e474256f72d3881556672e0df3663c25b2db89ef8bb5592c0fb093f8f
SMM	libraries/SafeMath.sol	bdb6d842f61551d03e01294740b196e965f156b898e90edde03897ed356c99d3

Understanding

External Dependencies

The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

There are a few dependent injection contracts or addresses in the current project:

- `WADA` for the contract `Router01` and `Router02`.
- `ICallee` for the contract `Pair`

We assume these contracts or addresses are valid and non-vulnerable actors and implement proper logic to collaborate with the current project.

Privileged Roles

The following roles are adopted to enforce the access control:

- Role `_owner` is adopted to update configurations of the contract `Collector`.
- Role `feeToSetter` is adopted to update configurations of the contract `Factory`.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

Findings

Critical	**0** (0.00%)
Major	**2** (15.38%)
Medium	**1** (7.69%)
Minor	**2** (15.38%)
Informational	**8** (61.54%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
CMO-01	Issues With The `convert()` Function From SushiSwap	Logical Issue	Medium	Resolved
CMO-02	Potential Sandwich Attacks	Logical Issue	Minor	Acknowledged
CMO-03	Missing Validation for Array Length	Logical Issue	Informational	Resolved
CMO-04	Unused imports	Coding Style	Informational	Resolved
ERC-01	Potential Risk On `approve()`/`transferFrom()` Methods	Logical Issue	Minor	Acknowledged
MOA-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
MOA-02	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
MOA-03	Proper Usage of `require` And `assert` Functions	Coding Style	Informational	Acknowledged
MOA-04	Variables That Could Be Declared as Immutable	Gas Optimization	Informational	Acknowledged
MOA-05	Missing Emit Events	Coding Style	Informational	Acknowledged
MOA-06	Lack of Input Validation	Volatile Code	Informational	Acknowledged
PMO-01	Divide by Zero	Logical Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
PTM-01	Initial Token Distribution	**Centralization / Privilege**	**Major**	Acknowledged

CMO-01 | Issues With The `convert()` Function From SushiSwap

Category	Severity	Location	Status
Logical Issue	Medium	Collector.sol: 107~128	Resolved

Description

A known exploit exists within the `Collector.convert()` function (inherits from SushiSwap), allowing an external attacker to steal funds within the `Collector` contract.

Example,

Suppose there is a [DIGG,WBTC] pair, which will generate DIGG-WBTC LP tokens as the fee to the `Collector` contract. The DIGG-WBTC LP tokens will be converted to `PToken` via the `convert()` function. However, there is no direct swap pair [DIGG,wada]. By default, these two tokens will be swapped into wada first as there is no “bridge” is set.

- The attacker creates a [DIGG,wada] swapping pair and add a small amount of liquidity.
- Transactions in the DIGG-WBTC pool will send fees (in the form of DIGG-WBTC LP token) to `Collector`.
- The attacker calls `convert(WBTC, DIGG)` to convert those fees to `PToken`
- Based on the code implementation, the call stack is:
  - Burn DIGG-WBTC LP token to get WBTC and DIGG
  - Swap WBTC/DIGG to wada first (due to no designated “bridge”)
  - Swap wada to `PToken`
- However, due to the pair [DIGG,wada] is created by the attacker with low liquidity. The swap transaction (4.b) will increase the price of wada significantly due to slippage in the new pair [DIGG, wada].
- Therefore, the attacker can use a small amount of wada to swap for a large amount of DIGG.

Additionally, another documented issue is that if the `stakingContract` (where the fees are transferred to) is forked from SushiBar, an exploiter can add lots of `PToken` to the `stakingContract`, run `convert()` and then remove the `PToken`. In this case, the attacker is able to withdraw a great amount of the fee.

These exploits are generally possible during the early stage of the project when the corresponding liquidity pools have low liquidity.

Recommendation

The current implementation enforces protection against flashloans, this is however not enough because a whale could perform this attack. Also, this attack could be performed at the early stages of the project.

In the short term, regularly call the `convert()` function and ensure corresponding pools are set up with a certain amount of liquidity.

In the long term, it is recommended to add access controls over the `convert()` function so only the team can call it.

Alleviation

[OccamX]: The team resolved this issue by adding access controls over the `convert()` function in commit [562e76287cc94596fedcd94e7c2fa0eaf1b691e3]

CMO-02 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue	Minor	Collector.sol: 201	Acknowledged

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction being attacked) a transaction to sell the asset.

The function `_swap()` directly interacts with UniswapV2Pair and doesn't set restrictions on slippage or minimum output amount, so transactions triggering this function are vulnerable to sandwich attacks, especially when the input amount is large.

Recommendation

In the short term, triggering `convert()` function regularly to avoid swapping large amount of tokens.

In the long term, add restrictions for slippage.

Alleviation

[OccamX]: The team acknowledged this and will follow the suggestion to call `convert()` function regularly.

CMO-03 | Missing Validation For Array Length

Category	Severity	Location	Status
Logical Issue	Informational	Collector.sol: 96~105	Resolved

Description

In `convertMultiple` function, the length of `token0` should be the same as the length of `token1`.

Recommendation

Consider adding the validation:

```
require(token0.length == token1.length,"the length of the array is invalid");
```

Alleviation

[OccamX]: The team resolved this issue by adding validation in commit 6fc74cecd04ce6c2897f64a91643dd7d93d08bee

CMO-04 | Unused Imports

Category	Severity	Location	Status
Coding Style	Informational	Collector.sol: 9	Resolved

Description

The linked code contains unused imports.

```
9  import "./interfaces/IERC20Permit.sol";
```

Recommendation

Remove the unused imports for simplicity and better code readability.

Alleviation

[OccamX]: The team resolved this issue by removing the unused imports in commit ad7ffd43058ae1f1a61032cce23cec13bd8ee6ed

ERC-01 | Potential Risk On `approve()` / `transferFrom()` Methods

Category	Severity	Location	Status
Logical Issue	Minor	ERC20Permit.sol: 63~66, 73~79	Acknowledged

Description

The `approve` function could be used in a Front-Running attack that allows a spender to transfer more tokens than the owner of the tokens ever wanted to allow the spender to transfer.

Here is a possible attack scenario:

- Alice allows Bob to transfer N of Alice's tokens (N>0) by calling `approve` method on the Token smart contract passing Bob's address and N as method arguments.
- After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls `approve` method again, this time passing Bob's address and M as the method arguments.
- Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls `transferFrom` method to transfer N Alice's tokens somewhere.
- If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain the ability to transfer another M tokens.
- Before Alice noticed that something went wrong, Bob calls `transferFrom` method again, this time to transfer M Alice's tokens.

So, Alice's attempt to change Bob's allowance from N to M (N>0 and M>0) made it possible for Bob to transfer N+M of Alice's tokens, while Alice never wanted to allow so many of her tokens to be transferred by Bob.

Reference:
An Attack Vector on Approve/TransferFrom Methods

Recommendation

We advise the client to use functions like `increaseAllowance()` and `decreaseAllowance()` from the `ERC20.sol` contract from OpenZeppelin.

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

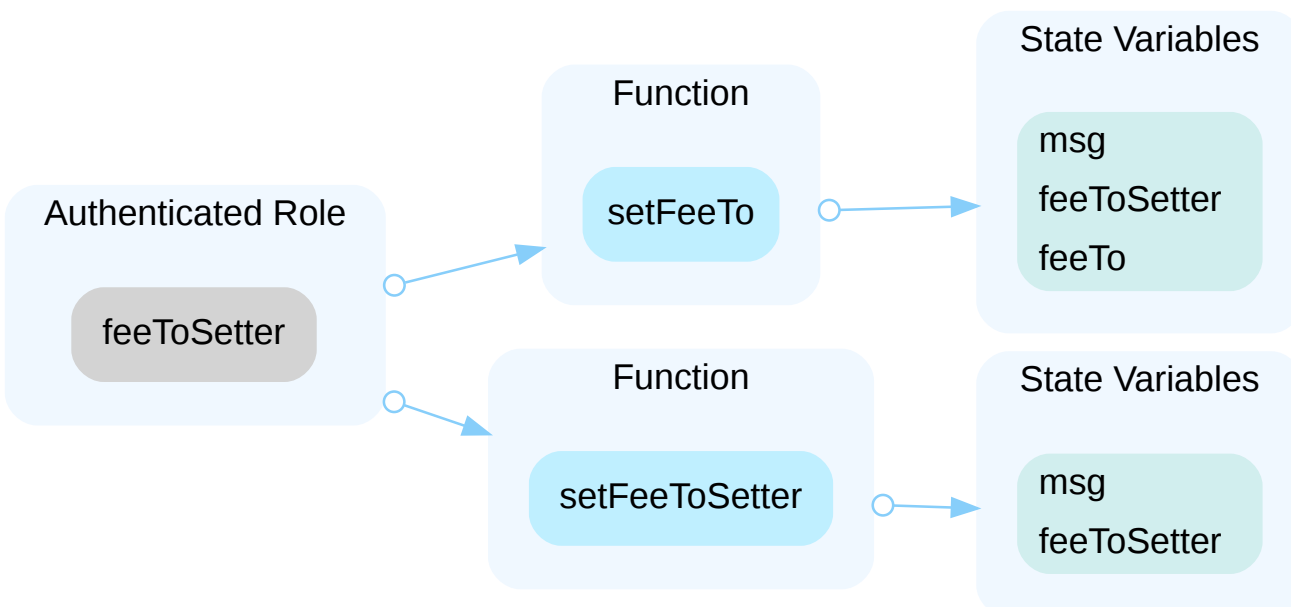
MOA-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Factory.sol: 40~43, 45~48 Collector.sol: 57~67, 69~71, 73~75, 88~90 Ownable.sol: 30~42	Acknowledged

Description

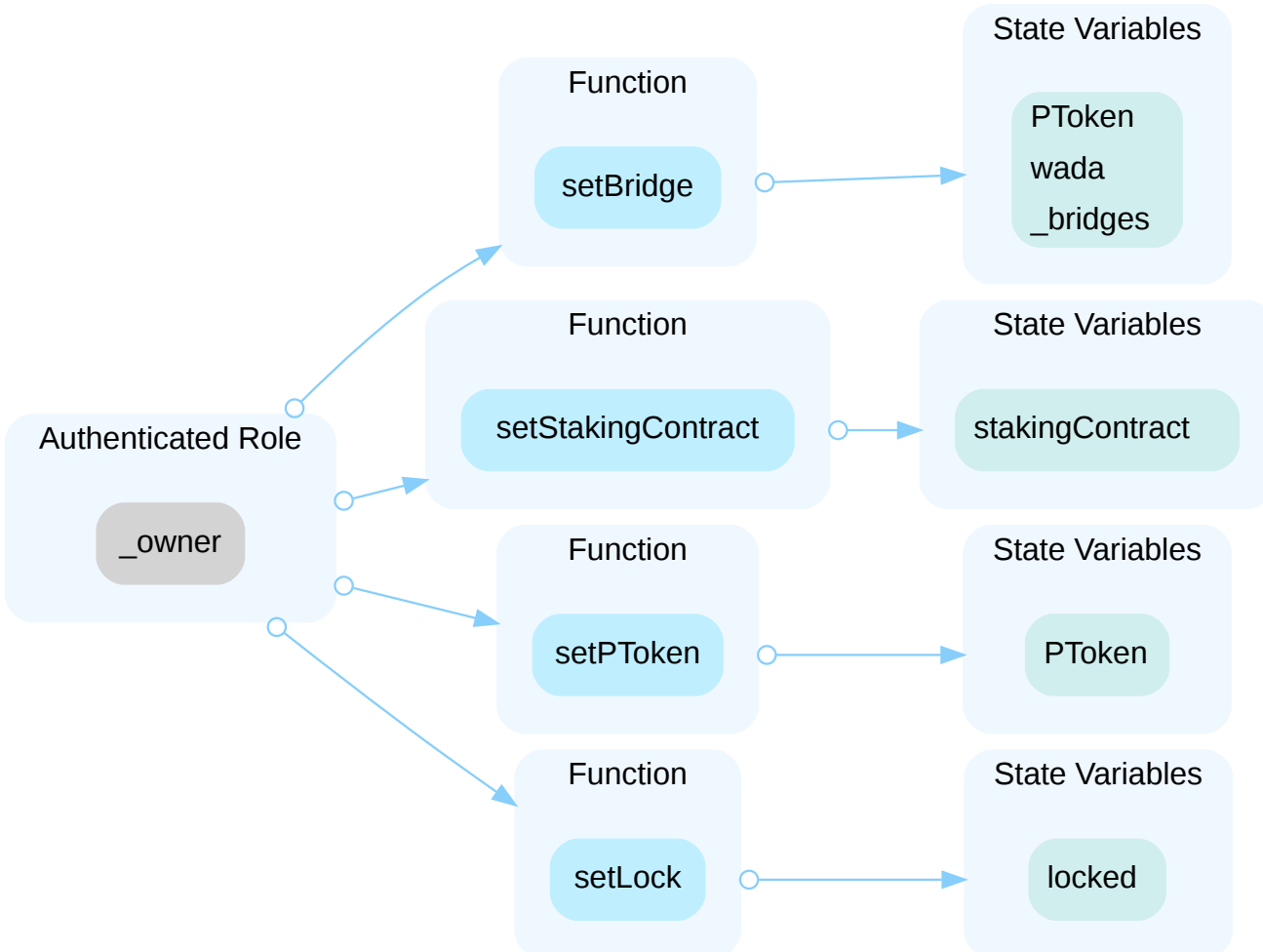
In the contract `Factory` the role `feeToSetter` has authority over the functions shown in the diagram below.

Any compromise to the `feeToSetter` account may allow the hacker to take advantage of this authority.



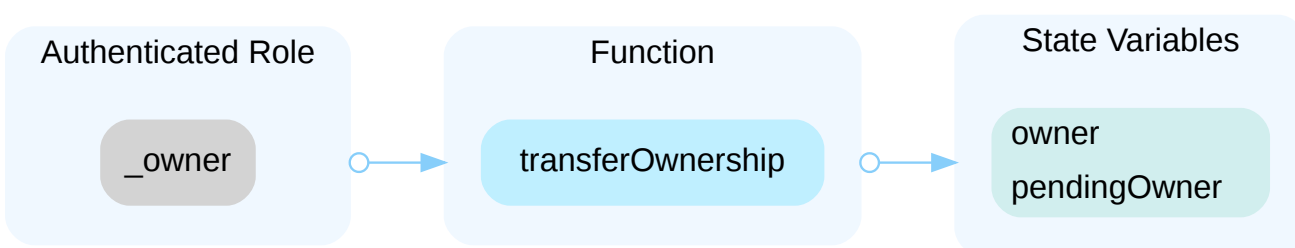
In the contract `Collector` the role `_owner` has authority over the functions shown in the diagram below.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



In the contract `Ownable` the role `_owner` has authority over the functions shown in the diagram below.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be

improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[OccamX]: The team will move these admin functionalities to multisigs in the future.

MOA-02 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Factory.sol: 1 ERC20Permit.sol: 1 CalHash.sol: 1 Collector.sol: 4 Router02.sol: 1 Pair.sol: 1 Router01.sol: 1 ProtocolToken.sol: 2 Ownable.sol: 5	Acknowledged

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to different compiler versions. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.0` the contract should contain the following line:

```
pragma solidity 0.6.0;
```

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

MOA-03 | Proper Usage Of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	Informational	Router02.sol: 29, 56, 97, 265, 314, 372 Router01.sol: 26, 52, 93, 214, 256	Acknowledged

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function.

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

MOA-04 | Variables That Could Be Declared As Immutable

Category	Severity	Location	Status
Gas Optimization	Informational	ERC20Permit.sol: 16 Pair.sol: 18	Acknowledged

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

MOA-05 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	Factory.sol: 40~43, 45~48 Collector.sol: 69~71, 73~75, 88~90 Pair.sol: 66~70	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications. For example,

- `Collector.setStakingContract()`
- `Collector.setPToken()`
- `Collector.setLock()`
- `Factory.setFeeTo()`
- `Factory.setFeeToSetter()`
- `Pair.initialize()`

Recommendation

Consider adding events for sensitive actions, and emit them in the functions.

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

MOA-06 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	Factory.sol: 15~17 Router02.sol: 23~26 Collector.sol: 39~48	Acknowledged

Description

In contract `Factory`, the assigned value to address type variable `_feeToSetter` should be verified as a non-zero value to prevent error.

In contract `Router02`, the assigned values to address type variables `factory`, `WADA` should be verified as non-zero values to prevent error.

In contract `Collector`, the assigned values to address type variables `factory`, `PToken`, and `wada` should be verified as non-zero values to prevent error.

Recommendation

In contract `Factory`, consider checking that the address is not zero in the function as shown below:

```
require(_feeToSetter != address(0),"_feeToSetter is zero address!");
```

In contract `Router02`, consider checking that the addresses are not zero in the constructor, like below:

```
require(_factory != address(0),"_factory is zero address!");
require(_WADA != address(0),"_WADA is zero address!");
```

In contract `Collector`, consider checking that the addresses are not zero in the constructor, like below:

```
require(_factory != address(0),"_factory is zero address!");
require(_PToken != address(0),"_PToken is zero address!");
require(_wada != address(0),"_wada is zero address!");
```

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

PMO-01 | Divide By Zero

Category	Severity	Location	Status
Logical Issue	Informational	Pair.sol: 143~145	Acknowledged

Description

If the value of `totalSupply` is 0, the following two division operations will fail due to the divide by 0 error, which ultimately make the invocation to `burn()` function fail.

```
144  amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
145  amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
```

Recommendation

Consider add the following validation in the function `burn()`

```
1  require(totalSupply != 0, "The value of totalSupply must not be 0");
```

Alleviation

[OccamX]: The team acknowledged this issue and decided not to change the current codebase at this stage.

PTM-01 | Initial Token Distribution

Category	Severity	Location	Status
Centralization / Privilege	**Major**	ProtocolToken.sol: 12~14	Acknowledged

Description

All of the tokens are sent to the `assetManager` when deploying the contract. This could be a centralization risk as the `assetManager` can distribute tokens without obtaining the consensus of the community.

```
12     constructor(uint totalSupply, address assetManager, string memory name, string
memory symbol) ERC20(name, symbol) ERC20Capped(totalSupply) {
13         _mint(assetManager, totalSupply);
14     }
```

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Alleviation

[OccamX]: The assetManager will be a multisig to ensure decentralization.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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.

Logical Issue

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

Volatile Code

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

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 but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

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 you (“Customer” or 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, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND

“AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

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.