CERTIK

Security Assessment

Mistswap

Nov 25th, 2021

Table of Contents

Summary

This report has been prepared for Mistswap to discover issues and vulnerabilities in the source code of the Mistswap 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	Mistswap
Description	DEX, Staking
Platform	smartBCH
Language	Solidity
Codebase	https://github.com/mistswapdex/mistswap/commit/f0155915743726804bf911b5ab246bcf33baf988
Commit	f0155915743726804bf911b5ab246bcf33baf988

Audit Summary

Delivery Date	Nov 25, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

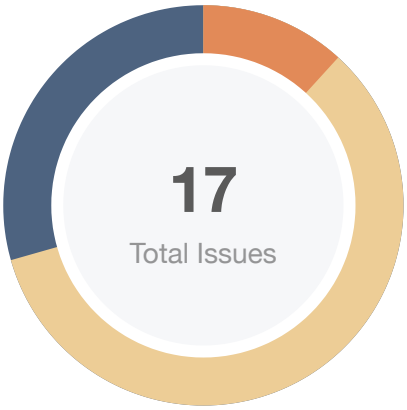
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	2	0	0	1	1	0
Medium	0	0	0	0	0	0
Minor	10	0	0	10	0	0
Informational	5	0	0	4	1	0
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
MCX	contracts/MasterChef.sol	2b612839c6ffc46ec24137790fff99e3c766b2004318e9e8d1fe0d69e8445529
MCV	contracts/MasterChefV2.sol	a02b948d831b821050c9999a852a7b4c848a315572af21beea73129a42079c59
MIG	contracts/Migrator.sol	d96a6c625a83433911dbcff078d17d3698284dcfa7ad0324e827ad45add8f7f5
MUL	contracts/Multicall2.sol	4f3038037239ac0548616d09ac24209779767e9e2ad9c68ed5a6537002b48805
OWN	contracts/Ownable.sol	a8eca79fa7812a24db2f0bd26fa9f31beccf2765b63313395ed04e16f1c1deda
SBX	contracts/SushiBar.sol	168704ff3ff937f611ba2a0dd24a54666370744efa6330bd3458e072bac93c12
SMX	contracts/SushiMaker.sol	f48e4da7847de5905a5d169c551bb5c551c787f1646d77dbc4ed2762ac293472
SMK	contracts/SushiMakerKashi.sol	b2aecd09b837625c24099321e775b697445e985f47f32df7f6439b8482ec34fe
SRX	contracts/SushiRoll.sol	1cabaa2feb6fb90034272e93538d2d67eab889cf59b58eaebe87dd97d7724814
STX	contracts/SushiToken.sol	66a744686424341c1747304d9bfb78021402af9570374036585a9fcfdd1e2deb

Findings



Critical	0 (0.00%)
Major	2 (11.76%)
Medium	0 (0.00%)
Minor	10 (58.82%)
Informational	5 (29.41%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
MIST-01	Centralization Risk	Centralization / Privilege	Major	Partially Resolved
MIST-02	Check Effect Interaction Pattern Violation	Logical Issue	Minor	Acknowledged
MIST-03	Lack of Event Emitting	Coding Style	Informational	Partially Resolved
MCV-01	`add()` Function Not Restricted	Logical Issue	Minor	Acknowledged
MCV-02	Incompatibility With Deflationary Tokens	Volatile Code	Minor	Acknowledged
MCV-03	Potential Loss of Pool Rewards	Logical Issue	Minor	Acknowledged
MCV-04	Third Party Dependencies	Volatile Code	Minor	Acknowledged
MCV-05	Lack of Explicit Pool Validity Checks	Logical Issue	Informational	Acknowledged
MCX-01	`add()` Function Not Restricted	Logical Issue	Minor	Acknowledged
MCX-02	Incompatibility With Deflationary Tokens	Volatile Code	Minor	Acknowledged
MCX-03	Potential Loss of Pool Rewards	Logical Issue	Minor	Acknowledged
MCX-04	Third Party Dependencies	Volatile Code	Minor	Acknowledged
MCX-05	Lack of Explicit Pool Validity Checks	Logical Issue	Informational	Acknowledged
OWN-01	PendingOwner Can Claim Ownership After Direct Ownership Transfer	Logical Issue	Informational	Acknowledged
SMK-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged

ID	Title	Category	Severity	Status
SRX-01	Proper Usage of `require` And `assert` Functions	Coding Style	Informational	Acknowledged
STX-01	Delegation Not Moved Along With `transfer()`	Logical Issue	Major	Acknowledged

MIST-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	Global	Partially Resolved

Description

In the contracts `MasterChef.sol` and `MasterChefV2.sol`, the role `Owner` has the authority over the following functions:

- `add()`
- `set()`
- `setMigrator()`

In the contracts `SushiMaker.sol` and `SushiMakerKashi.sol`, the role `Owner` has the authority over the following function:

- `setBridge()`

In the contract `SushiToken.sol`, the role `Owner` has the authority over the following function:

- `mint()`

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

Recommendation

We advise the client to carefully manage the `Owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[Mistswap team]: MasterChef ownership has been transferred to a timelock. MasterChefV2 is not used or deployed currently. Setting bridges for SushiMaker and SushiMakerKashi will be handled carefully. We will transfer these to timelock ownership in a future upgrade. SushiToken ownership has been transferred to MasterChef so mint is only able to be performed by normal operation of MistSwap. We have long term plan for adding governance of MistSwap.

MIST-02 | Check Effect Interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	Global	Acknowledged

Description

In `MasterChef.sol` and `MasterChefV2.sol`, the order of external call/transfer and storage manipulation should follow the check-effect-interaction pattern in the following functions:

- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`

Recommendation

We advise the client to check if storage manipulation is before the external call/transfer operation, or adding a `nonReentrant` modifier to these functions. For example, the following code snippet can be taken as reference for function `emergencyWithdraw()`:

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[msg.sender];
    uint256 userAmount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
    pool.lpToken.safeTransfer(address(msg.sender), userAmount);
    emit EmergencyWithdraw(msg.sender, _pid, userAmount);
}
```

Reference: https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Alleviation

[Mistswap team]: This will be resolved in a future version of MasterChef

MIST-03 | Lack of Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Global	Partially Resolved

Description

Functions that affect the status of sensitive variables should emit events as notifications to the community.

In the contract `MasterChef.sol`:

- `add()`
- `set()`
- `migrate()`
- `setMigrator()`
- `updatePool()`

In the contract `MasterChefV2.sol`:

- `migrate()`
- `setMigrator()`

In the contract `SushiBar.sol`:

- `enter()`
- `leave()`

Recommendation

We recommend adding events for sensitive actions in the aforementioned functions, and emit them in the functions.

Alleviation

[Mistswap team]: All MasterChef actions are performed using Timelock which does emit events. SushiBar.sol emits events for mint and burn

MCV-01 | `add()` Function Not Restricted

Category	Severity	Location	Status
Logical Issue	Minor	contracts/MasterChefV2.sol: 112~124	Acknowledged

Description

When the same LP token is added into a pool more than once in function `add()`, the total amount of reward in function `updatePool()` will be incorrectly calculated. The current implementation is relying on the operation correctness to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

We recommend adding the check for ensuring whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate. This can be done by using a mapping of `addresses` -> `booleans`, which can restrict the same address from being added twice.

Alleviation

[Mistswap team]: Our procedure for adding pools eliminates this potential error at a higher layer by keeping a list of all pools and not generating `add` calls for pools which already exist.

MCV-02 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	Minor	contracts/MasterChefV2.sol: 215	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction fee) in a MasterChef, only 90 tokens actually arrived in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

The MasterChef takes the pool token balance (the `lpSupply`) into account when calculating the users' reward. An attacker can repeat the process of deposit and withdraw to lower the token balance (`lpSupply`) in a deflationary token pool and cause the contract to increase the reward amount.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to regulate the set of pool tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[Mistswap team]: We will look to improve MasterChef to add support for deflationary tokens.

MCV-03 | Potential Loss of Pool Rewards

Category	Severity	Location	Status
Logical Issue	Minor	contracts/MasterChefV2.sol: 112~124, 131~136	Acknowledged

Description

In the `add()` and `set()` functions of `MasterChef.sol`, the flag '_withUpdate' determines if all the pools will be updated. In the `add()` and `set()` functions of `MasterChefV2.sol`, there is no `massUpdatePools()` or `updatePool()` function call. This might lead to the loss of pool reward for existing pools.

For illustration, assume we have only one pool with `pool.allocPoint == 50` and `totalAllocPoint == 50` at the beginning. Now we want to add another pool with `pool.allocPoint == 50`. There will be two scenarios on calculating the pool reward in `MasterChef.sol`,

Case 1: `_withUpdate` is set to `true`.

- distribute the reward and update the pool.
- add the given pool information

Case 2: `_withUpdate` is set to `false`.

- add the given pool information

(Note: While we focused on the `add()` function, both the `add()` and `set()` functions update `totalAllocPoint`, which is used in calculation of pool rewards in the function `updatePool()`

- In Case 1, reward for the first pool is updated in the call to `updatePool()` where `sushiReward = multiplier.mul(sushiPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`.
- In Case 2, an update `totalAllocPoint = totalAllocPoint.add(_allocPoint)` is done first. Then `updatePool()` calculates the reward for the first pool: `sushiReward = multiplier.mul(sushiPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`. Because the second pool is sharing rewards with the first one, the amount of reward for the first pool becomes half as much as that in the first case.

Recommendation

We advise the client to always update pool rewards before updating pool information via `add()` or `set()` function call.

Alleviation

[Mistswap team]: Our procedure for updating pools includes calling massUpdate function prior to updates to farms with the `add` or `set` function calls.

MCV-04 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	contracts/MasterChefV2.sol: 59, 115	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party `lpToken`, `rewarder`, etc.. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of `MasterChef.sol` and `MasterChefV2.sol` requires interaction with 3rd parties. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Mistswap team]: We are keeping updated on all relevant security bulletins related to Sushi.

MCV-05 | Lack of Explicit Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	contracts/MasterChefV2.sol: 131, 146, 160, 191, 210, 233, 255, 281, 313	Acknowledged

Description

The following functions do not have any sanity check of whether the given `_pid` or `pid` exists in the array `poolInfo`

- `set()`
- `migrate()`
- `pendingSushi()`
- `updatePool()`
- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`
- `harvest()`
- `withdrawAndHarvest()`

Recommendation

We recommend applying the following modifier to the aforementioned functions for `_pid` validity check.

```
modifier PoolValidation(uint256 _pid) {
    require (_pid < poolInfo.length , "Pool does not exist.") ;
    _;
}
```

Alleviation

[Mistswap team]: We will ensure that pid exists for all admin actions.

MCX-01 | `add()` Function Not Restricted

Category	Severity	Location	Status
Logical Issue	Minor	contracts/MasterChef.sol: 106~125	Acknowledged

Description

When the same LP token is added into a pool more than once in function `add()`, the total amount of reward in function `updatePool()` will be incorrectly calculated. The current implementation is relying on the operation correctness to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

We recommend adding the check for ensuring whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate. This can be done by using a mapping of `addresses` -> `booleans`, which can restrict the same address from being added twice.

Alleviation

[Mistswap team]: Our procedure for adding pools eliminates this potential error at a higher layer by keeping a list of all pools and not generating `add` calls for pools which already exist.

MCX-02 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	Minor	contracts/MasterChef.sol: 245~250	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction fee) in a MasterChef, only 90 tokens actually arrived in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

The MasterChef takes the pool token balance (the `lpSupply`) into account when calculating the users' reward. An attacker can repeat the process of deposit and withdraw to lower the token balance (`lpSupply`) in a deflationary token pool and cause the contract to increase the reward amount.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to regulate the set of pool tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[Mistswap team]: We will look to improve MasterChef to add support for deflationary tokens.

MCX-03 | Potential Loss of Pool Rewards

Category	Severity	Location	Status
Logical Issue	Minor	contracts/MasterChef.sol: 111, 133	Acknowledged

Description

In the `add()` and `set()` functions of `MasterChef.sol`, the flag '_withUpdate' determines if all the pools will be updated. In the `add()` and `set()` functions of `MasterChefV2.sol`, there is no `massUpdatePools()` or `updatePool()` function call. This might lead to the loss of pool reward for existing pools.

For illustration, assume we have only one pool with `pool.allocPoint == 50` and `totalAllocPoint == 50` at the beginning. Now we want to add another pool with `pool.allocPoint == 50`. There will be two scenarios on calculating the pool reward in `MasterChef.sol`,

Case 1: `_withUpdate` is set to `true`.

- distribute the reward and update the pool.
- add the given pool information

Case 2: `_withUpdate` is set to `false`.

- add the given pool information

(Note: While we focused on the `add()` function, both the `add()` and `set()` functions update `totalAllocPoint`, which is used in calculation of pool rewards in the function `updatePool()`

- In Case 1, reward for the first pool is updated in the call to `updatePool()` where `sushiReward = multiplier.mul(sushiPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`.
- In Case 2, an update `totalAllocPoint = totalAllocPoint.add(_allocPoint)` is done first. Then `updatePool()` calculates the reward for the first pool: `sushiReward = multiplier.mul(sushiPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`. Because the second pool is sharing rewards with the first one, the amount of reward for the first pool becomes half as much as that in the first case.

Recommendation

We advise the client to always update pool rewards before updating pool information via `add()` or `set()` function call.

Alleviation

[Mistswap team]: Our procedure for updating pools includes calling massUpdate function prior to updates to farms with the `add` or `set` function calls.

MCX-04 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	contracts/MasterChef.sol: 53, 119	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party `lpToken`, `rewarder`, etc.. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of `MasterChef.sol` and `MasterChefV2.sol` requires interaction with 3rd parties. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Mistswap team]: We are keeping updated on all relevant security bulletins related to Sushi.

MCX-05 | Lack of Explicit Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	contracts/MasterChef.sol: 128, 148, 178, 210, 234, 256, 273	Acknowledged

Description

The following functions do not have any sanity check of whether the given `_pid` or `pid` exists in the array `poolInfo`

- `set()`
- `migrate()`
- `pendingSushi()`
- `updatePool()`
- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`
- `harvest()`
- `withdrawAndHarvest()`

Recommendation

We recommend applying the following modifier to the aforementioned functions for `_pid` validity check.

```
modifier PoolValidation(uint256 _pid) {
    require (_pid < poolInfo.length , "Pool does not exist.") ;
    _;
}
```

Alleviation

[Mistswap team]: We will ensure that pid exists for all admin actions.

OWN-01 | PendingOwner Can Claim Ownership After Direct Ownership Transfer

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/Ownable.sol: 31~37, 46~56	ⓘ Acknowledged

Description

If `pendingOwner` is set and is not zero address, the `pendingOwner` can claim ownership any time even if the original owner has transferred ownership directly to a new owner. In other words, the new owner could be deprived of ownership, because `pendingOwner` is never reset to 0 when ownership is transferred to the new owner.

Recommendation

We recommend resetting the “pendingOwner” to “address(0)” after a direct ownership transfer.

Alleviation

[Mistswap team]: We will ensure pendingOwner is reset to 0 if ownership is changed with direct flag.

SMK-01 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	contracts/SushiMakerKashi.sol: 12, 22, 104, 107, 110, 111	Acknowledged

Description

The contract is serving as the underlying entity to interact with `BentoBoxWithdraw`, `KashiWithdrawFee`, etc.. The contract only included interface for these contracts and not the actual implementation. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets.

Recommendation

We understand that the business logic of `SushiMakerKashi.sol` requires interaction with 3rd parties. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Mistswap team]: We are keeping updated on all relevant security bulletins related to Sushi.

SRX-01 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	Informational	contracts/SushiRoll.sol: 135	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

[Mistswap team]: We are not using SushiRoll.

STX-01 | Delegation Not Moved Along With `transfer()`

Category	Severity	Location	Status
Logical Issue	Major	contracts/SushiToken.sol: 12	Acknowledged

Description

The voting power of delegation is not moved from token sender to token recipient along with the `transfer()` function call. Current implementation of the `transfer()` function is from the standard `ERC20` protocol and does not invoke `_moveDelegates()`.

Recommendation

We advise the client to consider adopting a specific implementation of the standard that has a `_moveDelegates()` logic called upon transferring tokens.

Reference: https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAM.sol#L108

Alleviation

[Mistswap team]: We are not and have no plan to use the delegation or voting ability within SushiToken.

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.

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.

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.