CERTIK

Code Security Assessment

xToken #2

Feb 4th, 2022

Table of Contents

Summary

This report has been prepared for xToken to discover issues and vulnerabilities in the source code of the xToken #2 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	xToken #2
Platform	other
Language	Solidity
Codebase	https://github.com/xtokenmarket/liquidity-mining-terminal
Commit	6acc45176594bb450534bfe718292e64c44fb3fc fce2ccd4fb7cb37d19a7b6caa75a809d3d99b8fe fd2bdf6ce1fd4c419f9af6cc6e1f2107fb9ac981

Audit Summary

Delivery Date	Feb 04, 2022
Audit Methodology	Static Analysis, Manual Review
Key Components	LMTerminal, xAssetCLR

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
● Critical	0	0	0	0	0	0	0
● Major	1	0	0	1	0	0	0
● Medium	1	0	0	0	0	0	1
● Minor	3	0	0	1	0	0	2
● Informational	4	0	0	0	0	0	4
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IER	interfaces/IERC20.sol	890f8db14d1ff2277ae15e987aa323014f2faebbfbc9ff72b05d70e545f84872
IEC	interfaces/IERC20Extended.sol	34ad2458c5a39f9bceefdb1ca05f67085fc21dde40e5a17546a44cf5b588aded
ILM	interfaces/ILMTerminal.sol	ab67998cb75e2827f70871e2efa5f716f67dd3adea5a7f2d270ef039e308c4a6
IRE	interfaces/IRewardEscrow.sol	97d0cfde76eb5de1858dffab5c1b7a9c82be97dcd49678e59ff2fba72a4fe752
ISC	interfaces/IStakedCLRToken.sol	fa0c80311db76af5fd02eb9081ccd2769ad6341e9ef3bb38293826124c2ffd64
ISR	interfaces/IStakingRewards.sol	ea8e693959566b72c8e29fd066f7ac9562341ffc9d0da06ef98aa90ff4f1c4bb
IAC	interfaces/IxAssetCLR.sol	6be10d9260e7e68cbed16c358c42b71633d5a530e41da55fb1941425f04a361d
ITM	interfaces/IxTokenManager.sol	605ea707590d675da5aef846571c21a09850ec0464042de11eea98d25b3e2917
ABD	libraries/ABDKMath64x64.sol	5dc5f5e007eb73ddeb00d915886cb2ee3b7eb9c6d19853039dc494a305186a45
ULC	libraries/UniswapLibrary.sol	6090f1652ca578d41cf75b61e67081b1ac68950086c7a23aa2655064394490fe
UCK	libraries/Utils.sol	7d896c729dcd14f5dee06a4d9df4f16d1ad5eed97c9da50d3b3badfa82ee7373
LMT	proxies/LMTerminalProxy.sol	e5e916eae78b71b6beab59b715c921e15c28b7da12615d5d983112ba38ab8bbf
SCL	proxies/StakedCLRTokenProxy.sol	cb26a2a14f3c94de9598c49626d17e7c825f6daee371ee8dafceea3d18344127
ACL	proxies/xAssetCLRProxy.sol	206d91fc77e7f48349d4635bc33d20dfda36d0d12267002ada203a0d3a7d3c93
SRP	staking/proxies/StakingRewardsProxy.sol	da1309b52dfd4de24cd856cbc1d4fc918b1e89b26418da6399b9f456e009883f
REC	staking/RewardEscrow.sol	0b6c5fbccc8c9c661629e6f2b2fed9f1f7d82f9927c9f2407e0e8ef34bbf4779
SRC	staking/StakingRewards.sol	c5fdce8bef112bcc16e78b5ec78d89b9ef3d189e3a6794b3ff58050b4c4588b3
BLC	BlockLock.sol	01ec3c950ce065b61d5b328c38c1176047b996e2474af280d07a3325eddca1ad

ID	File	SHA256 Checksum
CLR	CLRDeployer.sol	b7215ee6f4de1a117f0966fe8a865fc316ee519a7f6e7d96caab33edb732885e
LMC	LMTerminal.sol	cf3a5ce2605c46f8cf7b2c7b5fa77dd1df827a1643dacdef4abfed1d39da969c
SCR	StakedCLRToken.sol	9c61e43aa084b7e869d7027e911b85c88dca5f1facbc8328e7b1d8e6599103e1
ACR	xAssetCLR.sol	fb5c3916ddb9617532bdb61ee452e70d9f0449fdbf79383a088e1ec6691877f0

Overview

xToken has created tokenized strategies that maximize yield, eliminate mental overhead, and maintain liquidity. These strategies are represented by ERC-20 tokens called **xAssets**.

In general, the **xToken** protocols build the generalized framework for liquidity mining on top of Uniswap V3.

The **xToken Terminal** allows users to deploy incentive liquidity pools with a concentrated price range. The pool admin is allowed to update the configurations of the pool, including vesting periods, reward emission rates, and reward duration.

External Dependencies

The contract is serving as the underlying entity to interact with third-party **Uniswap V3** protocols. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness.

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

- `clrImplementation`, `sCLRTokenImplementation` for contract `CLRDeployer`
- `clrDeployer`, `xTokenManager`, `rewardEscrow`, `uniswapFactory`, `positionManager`, and `uniContracts` for contract `LMTerminal`
- `clrPool` for contract `StakedCLRToken`
- `token0`, `token1`, `stakedToken`, `uniContracts`, `uniswapPool`, `manager`, and `terminal` for contract `xAssetCLR`
- `rewardTokens` for contract `RewardEscrow`
- `rewardTokens`, and `rewardEscrow` for contract `StakingRewards`

We assume these vulnerable actors are implementing proper logic to collaborate with the current project.

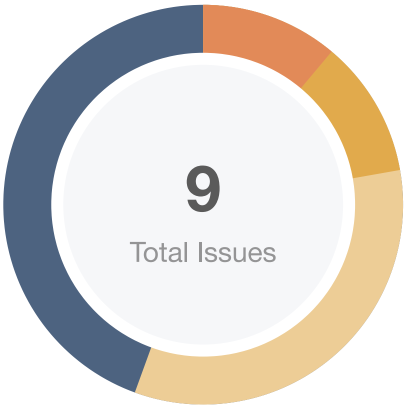
Privileged Roles

To set up the project correctly, improve overall project quality and preserve upgradability, the following roles are adopted in the codebase:

- The `owner` role is adopted in contract `CLRDeployer` to set the address of `clrImplementation` and `sCLRTokenImplementation`.
- The `CLRPool` role is adopted in contract `StakedCLRToken` to mint and burn `CLRTokens`.
- The `RevenueController` role is adopted in contract `LMTerminal` to withdraw fees from the contract.
- The `Terminal` role is adopted in contract `xAssetCLR` to mint, burn token and update configurations.
- The `owner` and `manager` roles are adopted in contract `xAssetCLR` to collect fees, update configurations, and pause/unpause the contract.

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 the `Timelock` contract.

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**1** (11.11%)
🟨 **Medium**	**1** (11.11%)
🟨 **Minor**	**3** (33.33%)
🟦 **Informational**	**4** (44.44%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
ACR-01	Potential Underflow	Mathematical Operations	● Minor	✓ Resolved
ACR-02	Lack of Error Message	Coding Style	● Informational	✓ Resolved
CKP-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
CKP-02	Lack of Event Emissions for Significant Transactions	Coding Style	● Informational	✓ Resolved
CKP-03	Improper Usage of `public` and `external` Type	Gas Optimization	● Informational	✓ Resolved
LMC-01	Potential Denial-of-Service of `withdrawFees()` Function	Logical Issue	● Medium	✓ Resolved
LMC-02	Incompatibility With Deflationary Tokens	Volatile Code	● Minor	ⓘ Acknowledged
REC-01	Gas Consumption on `rewardTokens`	Gas Optimization	● Minor	✓ Resolved
SCR-01	Lack of Error Message	Coding Style	● Informational	✓ Resolved

ACR-01 | Potential Underflow

Category	Severity	Location	Status
Mathematical Operations	● Minor	xAssetCLR.sol: 109, 111	✓ Resolved

Description

The linked statements apply a basic subtraction operation.

```
109                 10**(TOKEN_DECIMAL_REPRESENTATION - token0Decimals);
```

```
111                 10**(TOKEN_DECIMAL_REPRESENTATION - token1Decimals);
```

`TOKEN_DECIMAL_REPRESENTATION` is a `constant` integer with a value of 18. `token0Decimals`/`token1Decimals` represent the decimals of the input `_token0` and `_token1`.

In normal cases, underflow will not happen since most of tokens' decimals are under 18.

However, considering the input `_token0` and `_token1` are injected dependencies, the exact decimals of the given tokens are unknown.

For example, an input token with a "decimal" to be 20 could cause the underflow of the aforementioned arithmetic operation.

Recommendation

In the short term, apply "SafeMath" to the aforementioned arithmetic operations.

In the long term, regulate the set of tokens supported and add necessary mitigation mechanisms if there is a need to support tokens with decimal bigger than 18 should be accepted.

Alleviation

[xToken]: The team heeded the advice and resolved this issue by applying SafeMath in the commit fce2ccd4fb7cb37d19a7b6caa75a809d3d99b8fe.

ACR-02 | Lack Of Error Message

Category	Severity	Location	Status
Coding Style	● Informational	xAssetCLR.sol: 157, 193, 404, 405, 599	✓ Resolved

Description

The `require` statements can be used to check for conditions and throw an exception if the condition is not met, in which case the descriptive error messages provided by the developer will appear and help to track error and debug.

Recommendation

We recommend adding the error messages for the `require` statements on the aforementioned lines.

Alleviation

[xToken]: As the contract size is near the deployment limit, the team has cut out string messages in some places to lower it.

[CeriK]: This finding is related to coding style and will not cause any security issue.

CKP-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	LMTerminal.sol: 432, 459 CLRDeployer.sol: 41, 45 StakedCLRToken.sol: 41, 56 xAssetCLR.sol: 152, 192, 782, 794, 808, 377, 399, 427, 460, 477, 595, 614, 619 staking/RewardEscrow.sol: 64, 72, 80, 93, 110, 285	ⓘ Acknowledged

Description

In contract `LMTerminal`, the `RevenueController` role has the authority over the following functions:

- `LMTerminal.withdrawFees()`: Withdraw the ETH and reward token fees from the contract.
- `LMTerminal.withdrawClaimFees()`: Withdraw the claim fees for a given CLR pool.

In contract `xAssetCLR`, the `owner` and `manager` roles have the authority over the following functions:

- `xAssetCLR.collect()`: Collect the fees generated from the Uniswap pool.
- `xAssetCLR.mintInitial()`: Mint `CLR` tokens used for initializating the pool position.
- `xAssetCLR.rebalance()`: Rebalance the assets staked in Uniswap contract.
- `xAssetCLR.adminStake()`: Stake assets in the Uniswap contract.
- `xAssetCLR.adminSwap()`: Swap the LP tokens.
- `xAssetCLR.withdrawToken()`: Withdraw all the tokens in this contract to an arbitrary receiver.
- `xAssetCLR.pauseContract()`: Pause the contract.
- `xAssetCLR.unpauseContract()`: Unpause the contract.

In addition, the `Terminal` role has the authority over the following functions:

- `xAssetCLR.mint()`: Take the input of `token0` and `token1` and mint the `xAssetCLR` tokens.
- `xAssetCLR.burn()`: Burn the `xAssetCLR` tokens and return `token0` and `token1` assets.
- `xAssetCLR.setRewardsDuration()`: Set the duration of rewarding.
- `xAssetCLR.setRewardsAreEscrowed()`: Set whether rewards are escrowed or not.
- `xAssetCLR.initializeReward()`: Initialize the rewards with a given reward amount.

In contract `CLRDeployer`, the `owner` role has the authority over the following functions:

- `CLRDeployer.setCLRImplementation()`: Set the address of `clrImplementation`.
- `CLRDeployer.setsCLRTokenImplementation()`: Set the address of `sCLRTokenImplementation`.

In contract `StakedCLRToken`, the `CLRPool` role has the authority over the following functions:

- `StakedCLRToken.mint()`: Mint an arbitrary amount of tokens to an arbitrary recipient.
- `StakedCLRToken.burnFrom()`: Burn an arbitrary amount of tokens from any token holders.

In contract `RewardEscrow`, the `owner` role has the authority over the following functions:

- `RewardEscrow.addRewardsContract()`: Add a rewards contract address.
- `RewardEscrow.removeRewardsContract()`: Remove a rewards contract address.
- `RewardEscrow.addRewardsToken()`: Add a reward token contract address.
- `RewardEscrow.removeRewardsToken()`: Remove a reward token contract address.
- `RewardEscrow.setCLRPoolVestingPeriod()`: Set the vesting period for a given CLR pool.

Considering the owner of `RewardEscrow` contract should be `LMTerminal` contract, it will not cause any actual problem to the current project. This serves as a note to the `RewardEscrow` contract alone.

In addition, the `RewardsContract` role has the authority over the following functions:

- `RewardEscrow.appendVestingEntry()`: Add a new vesting entry at a given time and quantity to an account's schedule.

Any compromise to any account with any privileged role may allow the hacker to take advantage of this and disrupt operations involving this contract.

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., multi-signature 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.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[xToken]: The team acknowleadged this issue and decided not to change the current codebase.

CKP-02 | Lack Of Event Emissions For Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	staking/RewardEscrow.sol: 110 CLRDeployer.sol: 41, 45 xAssetCLR.sol: 152, 192, 399, 460, 477, 595, 614, 619, 782, 794, 808 StakedCLRToken.sol: 41, 56	✓ Resolved

Description

The following functions that affect the status of sensitive variables should emit events for better tracking contract status:

- `RewardEscrow.setCLRPoolVestingPeriod()`
- `CLRDeployer.setCLRImplementation()`
- `CLRDeployer.setsCLRTokenImplementation()`
- `xAssetCLR.mint()`
- `xAssetCLR.burn()`
- `xAssetCLR.mintInitial()`
- `xAssetCLR.adminStake()`
- `xAssetCLR.adminSwap()`
- `xAssetCLR.withdrawToken()`
- `xAssetCLR.pauseContract()`
- `xAssetCLR.unpauseContract()`
- `xAssetCLR.setRewardsDuration()`
- `xAssetCLR.setRewardsAreEscrowed()`
- `xAssetCLR.initializeReward()`
- `StakedCLRToken.mint()`
- `StakedCLRToken.burnFrom()`

Recommendation

We recommend adding event emissions for the sensitive actions in the aforementioned functions.

Alleviation

[xToken]: The team heeded the advice and added events for some of significant transactions in the commit fce2ccd4fb7cb37d19a7b6caa75a809d3d99b8fe.

As the contract size is too big, the team has cut out events where possible.

For StakedCLRToken we have corresponding Transfer events for mint and burn from the ERC-20 OpenZeppelin implementation, so it's not necessary to add events there.

CKP-03 | Improper Usage Of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	● Informational	LMTerminal.sol: 432, 459 staking/StakingRewards.sol: 145, 152, 133 staking/RewardEscrow.sol: 245, 268, 208, 285 CLRDeployer.sol: 22, 30, 45, 41 xAssetCLR.sol: 226, 377	✓ Resolved

Description

`Public` functions that are never called by the contract could be declared as `external`. `External` functions are more efficient than `public` functions.

Recommendation

We recommend using the external attribute for public functions that are never called within the contract.

Alleviation

[xToken]: The team heeded the advice and resolved this issue in the commit fce2ccd4fb7cb37d19a7b6caa75a809d3d99b8fe.

LMC-01 | Potential Denial-of-Service Of `withdrawFees()` Function

Category	Severity	Location	Status
Logical Issue	● Medium	LMTerminal.sol: 434	✓ Resolved

Description

In the contract `LMTerminal`, users can deploy Uniswap pools with given tokens by calling `deployIncentivizedPool()`, which leads to a new `clrPool` (i.e., a `IxAssetCLR` instance) added to the array `deployedCLRPools`.

```
268         deployedCLRPools.push(clrPool);
```

The revenue controller can withdraw xtoken fees via function `withdrawFees()`, which would iterate the array `deployedCLRPools` (i.e., deployed CLR pools).

```
434         for (uint256 i = 0; i < deployedCLRPools.length; ++i) {
435             address[] memory rewardTokens = deployedCLRPools[i]
436                 .getRewardTokens();
437 
438             for (uint256 j = 0; j < rewardTokens.length; ++j) {
439                 uint256 fees = rewardFeesTotal[rewardTokens[j]];
440                 IERC20(rewardTokens[j]).safeTransfer(msg.sender, fees);
441                 rewardFeesTotal[rewardTokens[j]] = 0;
442                 emit TokenFeeWithdraw(rewardTokens[j], fees);
443             }
444         }
```

Considering the gas cost of the function `withdrawFees()`, if the array `deployedCLRPools` is very large, calling `withdrawFees()` could fail due to an "out-of-gas" error.

An attacker can call `deployedCLRPools` repeatedly to expand the size of `deployedCLRPools` array, thus launching a DoS (Denial-of-Service) attack.

Recommendation

In the short term, calling `withdrawClaimFees()` to manually withdraw fees when `withdrawFees()` function failed.

In the long term, redesign the `withdrawFees()` logic. Instead of iterating all the elements in the `deployedCLRPools` array, it can iterate a given length and withdraw fees by segments.

Alleviation

[xToken]: The team heeded the advice and resolved this issue by refactoring the function `withdrawFees()` in the commit fd2bdf6ce1fd4c419f9af6cc6e1f2107fb9ac981.

LMC-02 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	● Minor	LMTerminal.sol: 240, 245	ⓘ 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, when a user calls `deployIncentivizedPool()` to deploy a new pool. A certain amount of `token0` and `token1` will be transferred to `LMTerminal` contract. Then, the received tokens will be further forwarded to `clrPool` (i.e., a `xAssetCLR` instance) in the `mintInitial()` call.

```
239         // Transfer initial mint tokens to Terminal
240         IERC20(pool.token0).safeTransferFrom(
241             msg.sender,
242             address(this),
243             pool.amount0
244         );
245         IERC20(pool.token1).safeTransferFrom(
246             msg.sender,
247             address(this),
248             pool.amount1
249         );
250
251         // Create Uniswap V3 Position, seed with initial liquidity
252         clrPool.mintInitial(pool.amount0, pool.amount1, msg.sender);
```

If `token0`/`token1` is a deflationary token (f.e., with a 10% transaction fee), only 90% of tokens actually arrived in the `LMTerminal` contract. Since it might not be enough tokens to be transferred to `clrPool`, it could cause calling `mintInitial()` to revert.

Recommendation

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

Alleviation

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

REC-01 | Gas Consumption On `rewardTokens`

Category	Severity	Location	Status
Gas Optimization	● Minor	staking/RewardEscrow.sol: 98	✓ Resolved

Description

The function `removeRewardsToken` is meant to delete a given reward token. However, the “delete” operation located at line 98 might not fulfill its intended purpose. The “delete” operation on arrays simply zero out the storage space and leave a gap, instead of reducing the size of the array and essentially removing the element.

This could lead to extra gas costs when iterating the array `rewardTokens`.

Recommendation

we advise to "pop" out the last element instead of "delete".

Example,

```
rewardTokens.pop();
```

Alleviation

[xToken]: The team heeded the advice and resolved this issue in the commit fce2ccd4fb7cb37d19a7b6caa75a809d3d99b8fe.

SCR-01 | Lack Of Error Message

Category	Severity	Location	Status
Coding Style	● Informational	StakedCLRToken.sol: 22	✓ Resolved

Description

The `require` statement can be used to check for conditions and throw an exception if the condition is not met, in which case the descriptive error message provided by the developer will appear and help to track error and debugging.

Recommendation

We recommend adding the error message for the `require` statement on the aforementioned line.

Alleviation

[xToken]: The team heeded the advice and resolved this issue by applying SafeMath in the commit fce2ccd4fb7cb37d19a7b6caa75a809d3d99b8fe.

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.

Mathematical Operations

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

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.

CERTIK