ACryptoS

Security Assessment

Mar 24th 2021

Summary

This report has been prepared for ACryptoS Finance smart contracts and strategies to discover issues and vulnerabilities in the source code as well as any 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 security experts.

The security assessment resulted in 8 findings that are informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest below recommendations that could better serve the project from the security perspective:

1. Enhance general coding practices for better structures of source codes;
2. Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
3. Provide more comments per each function for readability, especially contracts are verified in public;
4. Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Name	ACryptoS Finance
Codebase	https://github.com/acryptos/acryptos-protocol/blob/main/strategies/StrategyACryptoSCakeTokenTokenV2.sol
Commit Hash	82ebbede7e0bd0b480d424ccb48d05fb29128457

Engagement Summary

Delivery Date	Mar 24th, 2021
Methodology	Static analysis and manual review
Contracts in Scope	1
Contract - Token	StrategyACryptoSCakeTokenTokenV2.sol

Finding Summary

Total	8
Critical	0
Medium	0
Minor	0
Informational	8

Understandings

`StrategyACryptoSCakeTokenTokenV2` is a materialized representation of the typical strategy design pattern: Controlled by the vault and interacts with third party protocols to earn gainings in a more active way. Four tokens are involved:

1. `want`: normally the LP token of a corresponding protocol's liquidity pool (in this case, `pancakeswap`);
2. `cake`: the native protocol reward token of `pancakeswap`;
3. `tokenA`: Correlate with the pool it interacts with; bundle with tokenB for providing liquidity. For instance, `WBNB`;
4. `tokenB`: Correlate with the pool it interacts with; bundle with tokenA for providing liquidity. For instance, `XVS`.

There are four types of fees that would be commissioned out of the yields:

1. `performance`: 4.5% goes to the reward address of its vault the controller;
2. `strategist`: 0.5% goes to the designer of the strategy (an address set via constructor);
3. `withdrawal`: 0.5% goes to the reward address of its vault the controller;
4. `harvester`: 0.3% goes to the end user whoever spends the gas and triggers the harvest action.

The strategy itself is implemented at harvest() function and publicly invocable. It firstly stakes all the LP token into the pool; then uses the CAKE it gained (with commission cut) to swap into A then B (use half of the A); in the end bundle A&B to provide liquidity over the pool specified for future earnings like swap fee.

ACryptoS team is actively maintaining a detailed information page listing all the contracts within its protocol system, a live example could be referenced to better understand the logic and functionality of this smart contract:
https://bscscan.com/address/0x725462aaea2fea77185763b176bc9e2ffd659b0c

Findings

ID	Title	Severity	Response
CTK-ACS-1	Proper usage of public and external	Informational	Acknowledged
CTK-ACS-2	Function return Value Ignored	Informational	Acknowledged
CTK-ACS-3	Missing zero Address Validation	Informational	Acknowledged
CTK-ACS-4	Privileged governance	Informational	Acknowledged
CTK-ACS-5	Earlier require when withdraw	Informational	Acknowledged
CTK-ACS-6	Better code commenting	Informational	Acknowledged
CTK-ACS-7	Third party interaction & dependency	Informational	Acknowledged
CTK-ACS-8	Redundant interface definition	Informational	Acknowledged

CTK-ACS-1 | Proper usage of public and external

Type	Severity	Location
Gas Optimization	Informational	StrategyACryptoSCakeTokenTokenV2.sol: L109, L228

Description

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

StrategyACryptoSCakeTokenTokenV2.sol: `deposit()`, `harvest()`, `balanceOf()`

Recommendation

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

CTK-ACS-2 | Function return value ignored

Type	Severity	Location
Volatile Code	Informational	StrategyACryptoSCakeTokenTokenV2.sol: L183, L195

Description

Return values of function `swapExactTokensForTokens()` are ignored in function `_convertCakeToWant()`.

Recommendation

We advise developers to handle the return value of `swapExactTokensForTokens()` to follow the smart contract best practices.

CTK-ACS-3 | Missing zero address validation

Type	Severity	Location
Volatile Code	Informational	StrategyACryptoSCakeTokenTokenV2.sol: L255, L260, L265

Description

The assigned value to `governance`, `controller` and `strategist` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in `setGovernance()` `setController()` and `setStrategist()`. Violation of this may cause losing ownership of `governance`, `controller` and `strategist`.

Recommendation

Check that the address is not zero by adding checks in function `setGovernance()`, `setController()` and `setStrategist()`.

Alleviation

The team decided to leave as it is given the fact that address can still be set mistakenly to any other value resulting in loss of control with 0 checking added. Furthermore, these calls are behind 48H time lock and we have community timelock monitor and Telegram timelock monitor bot, adding several layers of verification to check if an erroneous call is made.

CTK-ACS-4 | Privileged governance

Type	Severity	Location
Business Model	Informational	StrategyACryptoSCakeTokenTokenV2.sol: L275, L280, L285

Description

The Governance of StrategyACryptoSCakeTokenTokenV2 has permission to update the parameters on rewards and set `controller`/`strategist` without obtaining the consensus of the community.

Recommendation

Migrate governance to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations. The team has maintained a detailed portal for the community to check governance and timelock information:

1. https://docs.acryptos.com/governance
2. https://docs.acryptos.com/security-and-risks

Alleviation

The team has responded that governance is behind 48H timelock, and there are community timelock monitor and Telegram timelock monitor bot:

1. Community timelock monitor: https://unrekt.net/acryptos/timelock.html
2. Telegram timelock monitor bot: https://t.me/acryptos9/59652

CTK-ACS-5 | Earlier require when withdraw

Type	Severity	Location
Coding Style	Informational	StrategyACryptoSCakeTokenTokenV2: L134

Description

When the amount provided in the withdraw() parameter is higher than the sum of current LP balance of the strategy and the staked LP in the cakeChef, the function would revert when calling _withdrawSome(_amount.sub(_balance)), which is correct but could be having this as a require statement at the beginning of the function.

Recommendation

The current logic is correct but may be slightly improved for better readability when reviewed by others. For instance:

```
require(CakeChef(cakeChef).userInfo(cakeChefPid,address(this)).add(IERC20(want).balanceOf(address(this))) >= amount)
```

CTK-ACS-6 | Better code commenting

Type	Severity	Location
Coding Style	Informational	StrategyACryptoSCakeTokenTokenV2

Description

The business logic of the strategy is well implemented, yet lack of code comments for the contract and core methods.

Recommendation

Provide more comments on functions for better readability and future maintainability.

CTK-ACS-7 | Third party interaction & dependency

Type	Severity	Location
Business Model	Informational	StrategyACryptoSCakeTokenTokenV2

Description

The strategy itself heavily relies on the interaction and well functioning of the pancakeswap protocol for potential gaining rewards.

Recommendation

The scope of the audit would treat those 3rd party entities as black boxes and assume its functional correctness. However in the real world, 3rd parties may be compromised that led to assets lost or stolen. The ACryptoS team has also provided a section to address similar topics there: https://docs.acryptos.com/security-and-risks

CTK-ACS-8 | Redundant interface definition

Type	Severity	Location
Coding Style	Informational	StrategyACryptoSCakeTokenTokenV2.sol: L12

Description

The source code lists all the function signatures of the CakeChef, however only four functions are actually invoked: `deposit`, `withdraw`, `emergencyWithdraw` and `userInfo`.

Recommendation

Though harmless from the runtime perspective, it is a good practice to keep only the necessary functions for the interface to improve the cleanliness and better readability.

Appendix | Finding Categories

Gas Optimization

Refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction in the total gas cost of a transaction.

Mathematical Operations

Refer to exhibits that relate to mishandling of math formulas, such as overflows, incorrect operations, etc.

Logical Issue

Refer to exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Business Model

Refer to contract or function logics that are debatable or not clearly implemented according to the design intentions.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

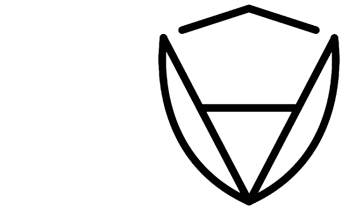
This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About CertiK

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

Provable Trust For All