CERTIK

CertiK Audit Report for Akropolis

Contents

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 Verification Services Agreement between CertiK and Akropolis (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”).

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK’s mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that the project is checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and verifications on the project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance’s BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor.

Executive Summary

Akropolis is a project that enables decentralized and autonomous communities. The Sparta Pool (which we have previously audited) is responsible for two things:

- Minting pool tokens by depositing liquid tokens
- Allowing people to pledge pool tokens to collateralize loans, and others to take out loans and pay them back

The current round was concerned with the audit of the Delphi Pools project. The main premise is to pool liquidity and reinvest it in various staking and savings protocols. The interest is then distributed to pool token holders. These pool tokens are very similar to the PTK token used for loan collateralization. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

After conducting our initial review, we identified two major, one minor, and a few information-level points. The team has successfully fixed all the major and minor points.

Testing Summary

SECURITY LEVEL

TIER-ONE

VERY HIGH CONFIDENCE

VERIFIED BY CERTIK

TYPE	Smart Contract Audit
SOURCE CODE	https://github.com/akropolisio/delphi-pools
PLATFORM	Ethereum Virtual Machine
LANGUAGE	Solidity
REQUEST DATE	August 12, 2020
FINAL DELIVERY DATE	September 7th, 2020
METHODS	A comprehensive examination has been performed using Whitebox Analysis. In detail, Dynamic Analysis, Static Analysis, and Manual Review were utilized.

Review Notes

Introduction

AkropolisOS is a protocol to enable autonomous and decentralized communities. We previously (June, July 2020) conducted a security audit of the entire smart contract codebase of AkropolisOS. This audit was focused on Delphi Pools, whose aim is to pool communities to together save and stake tokens. The initial audit was conducted from August 3, 2020 to August 7, 2020. The Client then added two more modules (BalancerProtocol and UniswapV2Protocol), and made corrections due to the first report. The goal of this audit was to ensure the security and correctness of the relevant parts of the codebase.

Scope of Work

The files that were in scope were those matching contracts/**/*, except contracts/modules/defi/(CurveFiYProtocol.sol|RAYProtocol.sol).

The exact version of the codebase of the first round of the audit was that corresponding to the commit:
d3043e2f76ee2cc2bb14493127b8970da925626f

For the second round, the commit was:
cbaf74be02ea39d523aa9c1cd2cf25c01db79b1b

For the final report, the commit was:
0e1c0611bb0743704ef53a328c7d113981624cfe

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Incorrect variable	Logic	Major	RewardDistributions.sol

Description:

On L171 in `RewardDistributions`, the last variable `totalAmount` should be amount instead. Otherwise we would be taking the cumulative value every time, while instead we want to use the current reward amount inside the loop body.

Recommendation:

Change `totalAmount` to `amount`.

Alleviations:

This issue has been fixed.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary check	Logic	Informational	RewardDistributions.sol

Description:

On L138, there is a check `_tokens.length > 0 && _amounts[0] > 0`. These variables come from the protocol wrapper, in particular `ProtocolBase.claimRewards` (none of the protocol instantiations override this method). While the output array is filled with non-zero values only (ProtocolBase L55), we would consider the code to be more resilient if only the first check is performed. As far as we can see, there will not be any bugs arising if one of the elements of `_amounts` is non-zero. Henceforth we feel it is a better call to omit it.

Recommendation:

Change `_tokens.length > 0 && _amounts[0] > 0` to `_tokens.length > 0` (omit the second check).

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
distributeYeld is vulnerable	Function visibility	Major	SavingsModule.sol

Description:

Let us back up a little. `DistributionToken.distribute` is a function that allows a minter to add to the `distributionAccumulator`, the number of tokens distributed in the next distribution. `SavingsModule` is a minter in the `PoolToken`. The function `SavingsModule.distributeYeldInternal` gets the current balance, and if it is larger than the previous balance, it distributes the difference. The precise problem is it doesn't update the previous balance - that is the role of `SavingsModule.updateProtocolBalance`. It follows these functions should be called in tandem, otherwise `distributeYeldInternal` could be called multiple times, leading to an inflation bug. This is precisely the case in `SavingsModule.distributeYeld`, which is a public function with no access control.

Recommendation:

Either restrict the visibility of this function, or add a jump to `SavingsModule.updateProtocolBalance` after every jump to `distributeYeldInternal`.

Alleviations:

The team opted to update to `ProtocolInfo.previousBalance` in `distributeYieldInternal`, hence the problem is resolved.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
De/normalized values	Logic	Minor	SavingsModule.sol

Description:

`SavingsModule.withdraw` accepts a denormalized value as its third parameter. We withdraw from the protocol and read the difference, which gives us a normalized value. On L184, we wish to normalize the parameter value in order to calculate the difference between the two: the fee. However, instead of a call to `normalizeTokenAmount`, there is a call to `denormalizeTokenAmount`.

Recommendation:

Change the function call on L184 from `denormalizeTokenAmount` to `normalizeTokenAmount`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Undefined variable	Compile error	Informational	DCAModule.sol

Description:

In `initialize()` on L113, there is an undeclared storage variable `rewardPool`, resulting in a compile error. We don’t think this variable is necessary.

Recommendation:

Remove lines 91 and 113 in DCAModule.sol.

Alleviations:

The team opted to remove the `DCAModule`.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Spelling errors	Language	Informational	Cross-module

Description:

We think there are a few spelling errors in the codebase. `cliamRewardsFromProtocol` is present in four files and `yeld` gives a total of thirteen occurrences.

Recommendation:

Change `cliamRewardsFromProtocol` to `claimRewardsFromProtocol` and `yeld` to `yield`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
`MAX_UINT` Approvals	Informational	Informational	Cross-module

Description:

There are multiple places where `MAX_UINT` is approved. While it is a trade-off between user experience and is up to the developers, we have to acknowledge that it does increase the potential consequences of a possible vulnerability.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Possible better Compound interaction	External interaction	Minor	CompoundProtocol.sol

Description:

`CompoundProtocol.cliamRewardsFromProtocol` calls `comptroller.claimComp`. According to the Comtroller documentation, this will loop through all markets and claim COMP gained in each. Furthermore, it will do for both borrowers and suppliers, while the current contract acts only as a supplier. This is likely to be inefficient gas-wise.

Recommendation:

The third function in the docs allows to specify both the `cTokens` to use as well as restrict to suppliers only. We recommend using that function.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
rewardBalances not updated	Logic	Major	ProtocolBase.sol

Description:

`rewardBalances` in `ProtocolBase` have two main interactions. The function `claimRewards` abstracts away the interaction to the inheriting contract, but returns the reward tokens and their corresponding reward amounts. The function `withdrawReward` then allows those withdrawals. The problem is the balances are not updated in the claim function.

Recommendation:

After the if clause on L44, add a storage update to `rewardBalances[rtkn] = newBalance`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Possible infinite loop	Logic	Major	RewardDistributions.sol

Description:

The loops in `rewardBalanceOf` and `_updateRewardBalance` are incorrect. If the user's share for that particular distribution is zero, we omit the arithmetic in order to save gas (L199 & L205). The problem is the loop counter is incremented after these lines, so in those cases we achieve an OOG exception due to an unbounded loop.

Recommendation:

Increment the loop counter before those lines of code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Gas savings for first time users	Gas optimization	Informational	RewardDistributions.sol

Description:

For first-time users, we update `rewardBalances[user].nextDistribution` by looping through all distributions. It might be possible to save gas for example by setting `rewardBalances[user].nextDistribution` to the next current distribution directly when a user joins the platform.

Recommendation:

Make a check in `_updateRewardBalances` whether `rewardBalances[user].nextDistribution` is zero, and if so, set it to `toDstribution`, without executing the loop. This works because the only way to acquire `PoolTokens` is through minting or transferring. Both ways call `RewardDistributions.poolTokenBalanceChanged`, which calls `_udpateRewardBalance`. Hence if a user held shares in any previous distribution, `nextDistribution` would not be zero.

Alleviations:

The team opted to change the codebase, by skipping previous distributions if the user is a new one.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Incorrect loop counter increment	Logic	Major	RewardDistributions.sol

Description:

`rewardBalanceOf` on L92 is used to compute the reward amounts for an array of reward tokens. It performs this by looping through all pool tokens, and finding the reward balance w.r.t. that (reward, pool) combination. The problem is the second loop uses a counter j, but increments the outer loop counter i.

Recommendation:

Change `i++` to `j++` on L96 in RewardDistributions.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Incorrect logic in `rewardBalanceOf`	Logic	Major	RewardDistributions.sol

Description:

`rewardBalanceOf` on L111 is used to compute the reward balance of a user w.r.t. a protocol. We first get the current reward balance, then add to it any yet unadded due to outstanding distributions. The problem is that Inside the loop, we compute `sh = rb.shares[d.poolToken]`. This means we're calculating the reward amounts for a particular token for all distributions, when in fact we are interested in those coming from a single protocol. Changing `d.poolToken` to the parameter `poolToken` would not be correct either, as then we would be computing reward shares with respect to a different pool token then that of the distribution. The correct way would be to skip that distribution if `d.poolToken != poolToken`.

Recommendation:

On L119, change sh == 0 to sh == 0 || poolToken != d.poolToken (other solutions are also of course possible).

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Gas optimization for `rewardBalanceOf` funtion	Gas	Informational	RewardDistributions.sol

Description:

The three functions `rewardBalanceOf` are `view` functions that are not used anywhere in the code. If the intention is to use them only for web3 calls, then gas optimization is not important. However, we still feel it is worthy to point out they could be more gas efficient. The theoretic boundary for this computation is O(dr), where d is number of distributions and r is number of reward tokens. If Exhibit 13 is corrected, then the complexity of `rewardBalanceOf` on L92 is O(rpd), where p is the number of protocols / pool tokens.

Recommendation:

If these functions are intended to be called on-chain, it would make sense to refactor them to achieve theoretic minimum complexity.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Function Naming Conventions	Coding Style	Informational	CurveFiProtocol.sol: L37, L38, L39

Description:

The `deposit_add_liquidity` , `deposit_remove_liquidity_imbalance` , `reward_rewardToken` functions of the `CurveFiProtocol` contract have names which are not in mixed case. Solidity has a naming convention that should be followed.

Reference the Solidity style guide regarding function names.

Recommendation:

Rename `deposit_add_liquidity` to `depositAddLiquidity`, `deposit_remove_liquidity_imbalance` to `depositRemoveLiquidityImbalance` and `reward_rewardToken` to `rewardRewardToken` or `rewardToken` .

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Unused `return` Value	Optimization	Informational	BalancerProtocol.sol: L52, L57, L78, L79, L98 CompoundProtocol.sol: L41, L52, L59 UniswapV2Protocol.sol: L36

Description:

The returned values from the linked functions remain unused throughout the codebase.

Recommendation:

Require the call to a specific function to succeed.

Example (for BalancerProtocol.sol L52):

require(IERC20(tkn).approve(address(bpt), MAX_UINT256));

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary `else-if` Block	Optimization	Informational	BalancerProtocol.sol: L207, L217, CompoundProtocol.sol: L115, L125 CurveFiProtocol.sol: L222, L233 SavingsModule.sol: L374, L385

Description:

The comparison conducted on the last `else-if` branch of the block can be safely removed, as this code branch should always execute if the program passes through the two other code branches.

Recommendation:

We advise the team to remove redundant code.

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	BalancerProtocol.sol: L43, L51, L152, L164 CurveFiProtocol.sol: L259, L270 ProtocolBase.sol: L54, L57 DistributionToken.sol: L34, L41, L54, L62, L70, L224 SavingsModule.sol: L104 RewardDistributions.sol: L147, L176, L183, L190, L204

Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendation:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Shadowed Variable	Optimization	Informational	BalancerProtocol.sol: L17 CompoundProtocol.sol: L18 CurveFiProtocol.sol: L18 UniswapV2Protocol.sol: L17

Description:

The `BalancerProtocol` , `CompoundProtocol` , `CurveFiProtocol` and `UniswapV2Protocol` contracts has a constant variable named `MAX_UINT256` which shadows the constant variable of the same named declared in the `ProtocolBase` contract that they inherits from.

While this issue does not compromise the system due to the constants having the same declared value.

Recommendation:

The declaration of `MAX_UINT256` can be safely removed from the `BalancerProtocol` , `CompoundProtocol` , `CurveFiProtocol` and `UniswapV2Protocol` contracts and the inherited declaration of `ProtocolBase.MAX_UINT256` will fall in place without having to make changes to other code.

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Special Variable Update	Optimization	Informational	SavingsModule.sol: L333, L365 DistributionToken.sol: L186, L224

Description:

The newer versions of Solidity (namely 0.7.0 onwards) deprecated the `now` keyword.

Recommendation:

We advise the team to consider changing `now` with `block.timestamp` .

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Potential For Re-Entrancy	Language Specific Issue & Optimization	Informational	CompoundProtocol.sol: L29-L37

Description:

The `CompoundProtocol.initialize` function has the potential for re-entrancy due to ignoring the Solidity ”Check Effects Interactions” pattern. While this issue will not lead to compromising the system, it will lead to gas exhaustion and should generally be avoided.

Reference the Solidity `Check Effects Interactions` pattern:

Recommendation:

Apply changes to all state variables before transfering from ERC-20 tokens. If the transfer from the ERC-20 token fails, the state variables will be reverted:

Example for CompoundProtocol.sol, L29-L37:

```
function initialize(
    address _pool,
    address _token,
    address _cToken,
    address _comptroller
) public initializer {
    ProtocolBase.initialize(_pool);
    baseToken = IERC20(_token);
    cToken = ICErc20(_cToken);
    decimals = ERC20Detailed(_token).decimals();
    comptroller = IComptroller(_comptroller);
    compToken = IERC20(comptroller.getCompAddress());
}
baseToken.safeApprove(_cToken, MAX_UINT256);
```

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version	Coding Style	Informational	All `pragma` Statements

Description:

The compiler version utilized throughout the project uses the `^` prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

Recommendation:

We advise that the compiler version is locked at version `0.5.12` or whichever Solidity version higher than that satisfies the requirements of the codebase as an unlocked compiler version can lead to discrepancies between compilations of the same source code due to compiler bugs and differences.

Alleviations:

The team opted to refactor the codebase according to our references in an upcoming update.