AKROPOLIS DELPHI STAKING MODULE SMART CONTRACT AUDIT

December 23, 2020

AKROPOLIS

MixBytes()

CONTENTS

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Akropolis (name of Client). If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 PROJECT OVERVIEW

Akropolis is a domain-specific protocol that allows users to save in groups and get access to cheap uncollateralized loans, pensions and other financial services. Akropolis tokenizes credit unions and allows user groups to get uncollateralized loans, provide capital to pools, and earn interest out of it.

1.3 SECURITY ASSESSMENT METHODOLOGY

At least 2 auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 "Blind" audit includes:
> Manual code study
> "Reverse" research and study of the architecture of the code based on the source code only

Stage goal:
Building an independent view of the project's architecture
Finding logical flaws

02 Checking the code against the checklist of known vulnerabilities includes:
> Manual code check for vulnerabilities from the company's internal checklist
> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the logic, architecture of the security model for compliance with the desired model, which includes:
> Detailed study of the project documentation
> Examining contracts tests
> Examining comments in code
> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of the reports from all auditors into one common interim report document
> Cross check: each auditor reviews the reports of the others
> Discussion of the found issues by the auditors
> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level
Provide the client with an interim report

05 Bug fixing & re-check.
> Client fixes or comments on every issue
> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.4 EXECUTIVE SUMMARY

The audited scope includes smart contracts set which are a part of delphi project's staking module. Deplhi is a project that helps users to create and manage savings account for participating in several pools. The staking module provides reward calculation and distribution logic and support several reward tokens for single account at the same time.

1.5 PROJECT DASHBOARD

Client	Akropolis
Audit name	Delphi Staking module
Initial version	86c3cbdf80e0d785be68e9b6b720395a3a91df8b
Final version	6bb16cf13c7fa73306e90ab753f9601c310fe1eb
SLOC	604
Date	2020-11-27 - 2020-12-23
Auditors engaged	2 auditors

FILES LISTING

IERC900.sol	IERC900.sol
StakingPool.sol	StakingPool.sol
StakingPoolBase.sol	StakingPoolBase.sol
RewardVestingModule.sol	RewardVestingModule.sol

FINDINGS SUMMARY

Level	Amount
Critical	0
Major	2
Warning	3
Comment	3

CONCLUSION

Smart contracts were audited and several suspicious places were spotted. During audit no critial issues were found, two issues were marked as major because they could lead to some undesired behavior. After working on reported findings all of them were resolved and contracts assumed as secure to use according to our security criteria.

2. FINDINGS REPORT

2.1 CRITICAL

Not Found

2.2 MAJOR

MJR-1	Potential claim from vesting lock
File	StakingPool.sol
Severity	Major
Status	Fixed at a54edb11

DESCRIPTION

At this line StakingPool.sol#L139 we have a `distributionAmount` calculation and it will fail if `actualBalance` would be less than `expectedBalance`, however at line below StakingPool.sol#L140 we are checking that `actualBalance` is more than `expectedBalance`, so according to this check it seems the contract should just skip distribution in case of a negative reward, but this check is unreachable in that case because `actualBalance.sub(expectedBalance)` will revert the transaction.

RECOMMENDATION

We recommend to move the `distributionAmount` calculation under `actualBalance > expectedBalance` condition.

MJR-2	Integer overflow and out of array bound issue
File	RewardVestingModule.sol
Severity	Major
Status	Fixed at 38c4305a

DESCRIPTION

There is the edge case when we handle integer overflow that leads to out of array bound issue at line RewardVestingModule.sol#L134
In case if `epochsLength` is `1` and `lastEpoch.end > block.timestamp` here RewardVestingModule.sol#L134 we got `i == 0`.

RECOMMENDATION

We recommend to add a check that `i > 0` before `i - 1` operation. For now, this issue only causes a transaction failure, but we strictly recommend fixing it because that can lead to more serious problems in combination with other potential issues in future.

2.3 WARNING

WRN-1	Unsafe arithmetic operation
File	StakingPoolBase.sol
Severity	Warning
Status	Fixed at f64d86a8

DESCRIPTION

At this line StakingPoolBase.sol#L340 used a raw sum operation without `SafeMath` is used that can potentially lead to overflow.

```
unstakeAllAmount =
unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
```

RECOMMENDATION

We recommend using openzeppelin's `SafeMath` library.

WRN-2	Potential issue with re-entrancy
File	StakingPoolBase.sol
Severity	Warning
Status	Fixed at 63858357

DESCRIPTION

Method `withdrawStake` performs an external contract call here StakingPoolBase.sol#L487

```
require(
  stakingToken.transfer(_msgSender(), _amount),
  "Unable to withdraw stake");
```

The contract updates the `personalStake.actualAmount` variable only after the transfer. This order can lead to re-entrancy if `stakingToken` does not have a standardized behaviour (e.g. some triggers).

RECOMMENDATION

We recommend to call transfer after the state updating or implementing a re-reentrancy guard. Even if for now `stakingToken` is safe it's so easy to introduce a bug while updates take place in future.

WRN-3	Wrong error message
File	StakingPoolBase.sol
Severity	Warning
Status	Fixed at fe1e4da1

DESCRIPTION

There is a wrong error message defined in the `StakingPoolBase.sol` contract at line StakingPoolBase.sol#L203

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
```

RECOMMENDATION

We recommend to replace the message with the correct one.

2.4 COMMENTS

CMT-1	Too implicit an external call
File	StakingPoolBase.sol
Severity	Comment
Status	Fixed at 53797539

DESCRIPTION

There is a `canStake` modifier defined at line StakingPoolBase.sol#L99 that performs an external contract call before the payload execution. This approach is too implicit and can lead to different issues (e.g. re-entrancy).

RECOMMENDATION

We recommend removing external calls and complex logic from modifiers.

CMT-2	Mixed code formatting
File	StakingPoolBase.sol StakingPool.sol
Severity	Comment
Status	Fixed at 6bb16cf1

DESCRIPTION

There are several places with different formats (e.g. indentation, spacing between operands):

- StakingPoolBase.sol#L121-L123
- StakingPool.sol#L87

RECOMMENDATION

We recommend using some linter to keep the same code style in the project.

CMT-3	Uncontrolled loop iterations amount
File	StakingPoolBase.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

There is a place with a full iteration through the array at line StakingPoolBase.sol#L270. As compared to other similar loops that function doesn't have any analogue to avoid the loop.

RECOMMENDATION

We recommend implementing another function to bypass the gas limit in case of a huge array.

CLIENT'S COMMENTARY

This loop is in a view function and it provides a fast way to get the total staked amount without knowing the internal logic of StakingPool. In case when a more sophisticated approach is needed, one can use getPersonalStakeActualAmounts(). Also, this contract is not very well suited for a huge number of stakes, and this is not a requirement for it. I don't think we need to fix this.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

 Ethereum

 Cosmos

 EOS

 Substrate

TECH STACK

 Python

 Solidity

 Rust

 C++

CONTACTS

 https://github.com/mixbytes/audits_public

 https://mixbytes.io/

 hello@mixbytes.io

 https://t.me/MixBytes

 https://twitter.com/mixbytes