CERTIK

Security Assessment

Zooswap

Apr 8th, 2021

Summary

This report has been prepared for Zooswap smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. 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.

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 given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Zooswap
Description	ZooSwap (zooswap.net) is a decentralized trading platform built on OKExChain, with liquidity mining and trade mining mechanism
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/zoo-swap/zooswap-core
Commits	1. 4a62cb407f7b5797c2014ada2c12421156f6d114 2. 188e6aabdf56ba77ecfcb479e7f8767ef66be330

Audit Summary

Delivery Date	Apr 08, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	

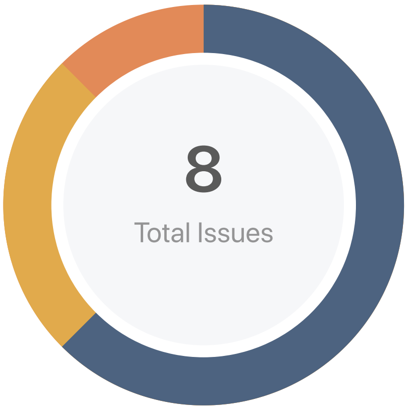
Vulnerability Summary

Total Issues	8
● Critical	0
● Major	1
● Minor	2
● Informational	5
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
HAZ	HalfAttenuationZooReward.sol	81d374f9c20c26f166eff86d3eb51017e8b7a307a55900b0af2bea24de41471d
ZKR	ZooKeeper.sol	deb5b4c0dd0af11a131e1721951ebdebf64790cf0b30348c40b812b88d803dfc
ZPK	ZooPark.sol	7d1b89ceb3b24900f4e22716edb0c68bbdf7aeb8e2535329e33c8f40fdfddd4c
ZRR	ZooRouter.sol	8fad20aec91a281857f5293f6a70080b1a0e1202fb56c9ceef98d51858932723
ZSM	ZooSwapMining.sol	a3c6ec71fdd864b76f1c93c1291b652b9b79ffb4aaaefa64319358e1cf1c7d64
ZTN	ZooToken.sol	d5df839d17ad4c8a522eda78eecbd9bbf310d5606e9d3eead9f9a306fabb2a93

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**1** (12.50%)
🟨 **Minor**	**2** (25.00%)
🟦 **Informational**	**5** (62.50%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
HAZ-1	Signed SafeMath Not Used	Mathematical Operations	● Informational	✓ Resolved
ZKR-1	Lack of Input Validation	Volatile Code	● Informational	✓ Resolved
ZPK-1	Potential Reentrant to Sensitive Functions	Logical Issue	● Minor	✓ Resolved
ZPK-2	Lack of Input Validation	Volatile Code	● Informational	✓ Resolved
ZSM-1	Signed SafeMath Not Used	Mathematical Operations	● Informational	✓ Resolved
ZSM-2	Potential Reentrant to Sensitive Functions	Logical Issue	● Minor	✓ Resolved
ZSM-3	Lack of Input Validation	Volatile Code	● Informational	✓ Resolved
ZSM-4	Calculation of Reward	Logical Issue	● Major	✓ Resolved

HAZ-1 | Signed SafeMath Not Used

Category	Severity	Location	Status
Mathematical Operations	● Informational	HalfAttenuationZooReward.sol: 63, 62	✓ Resolved

Description

Signed SafeMath from OpenZeppelin is not used making it possible for overflow/underflow, which will lead to an inaccurate log message.

Recommendation

Considering use OpenZeppelin's Signed SafeMath library function `mod` insteal of `%`

Considering use OpenZeppelin's Signed SafeMath library function `mul` insteal of `*` :

change

```
_to.sub(_startBlock) % _blockNumberOfHalfAttenuationCycle
```

to

```
_to.sub(_startBlock).mod( _blockNumberOfHalfAttenuationCycle)
```

change

```
_blockNumberOfHalfAttenuationCycle*2
```

to

```
_blockNumberOfHalfAttenuationCycle.mul(2)
```

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZKR-1 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	ZooKeeper.sol: 49~53	✓ Resolved

Description

The assigned value to `_devAddr` , `_investorAddr` , `_foundationAddr` and address of `_zoo` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in constructor of contract `ZooKeeper.sol` . Violation of this may cause losing ownership of `_devAddr` , `_investorAddr` , `_foundationAddr` and `_zoo` authorization.

Recommendation

Check that the address is not zero by adding following checks in the constructor of contract `ZooKeeper.sol` .

```
 require(_devAddr != address(0));
require(_investorAddr != address(0));
require(_foundationAddr != address(0));
require(address(_zoo) != address(0));
```

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZPK-1 | Potential Reentrant to Sensitive Functions

Category	Severity	Location	Status
Logical Issue	● Minor	ZooPark.sol: 179, 201, 218	✓ Resolved

Description

As token is a smart contract reference which can be arbitrarily assigned for once through add() in contract `ZooPark.sol` , it's implementation may be unknown to the user and potentialy includes logic to reentrant the sensitive functions such as `deposit()` , `withdraw()` and `emergencyWithdraw()` .

Recommendation

We advise developers to adopt `nonReentrant` modifier in openzeppelin to sensitive functions `deposit()` , `withdraw()` and `emergencyWithdraw()`

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZPK-2 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	ZooPark.sol: 77~78	✓ Resolved

Description

The address of `_zoo` and the address of `_zookeeper` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in the constructor of contract `ZooPark.sol` . Violation of this may cause losing ownership of `_zoo` , `_zookeeper` authorization.

Recommendation

Check that the address is not zero by adding following checks in the constructor of contract `ZooPark.sol` .

```
 require(address(_zoo) != address(0));
require(address(_zookeeper) != address(0));
```

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZSM-1 | Signed SafeMath Not Used

Category	Severity	Location	Status
Mathematical Operations	● Informational	ZooSwapMining.sol: 169	✓ Resolved

Description

Signed SafeMath from OpenZeppelin is not used making it possible for overflow/underflow, which will lead to an inaccurate log message.

Recommendation

Considering use OpenZeppelin's Signed SafeMath library function `add` instead of `+` :

change

```
total += user.amount.mul(accZooPerShare).div(1e12).sub(user.rewardDebt);
```

to

```
total = user.amount.mul(accZooPerShare).div(1e12).sub(user.rewardDebt).add(total);
```

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZSM-2 | Potential Reentrant to Sensitive Functions

Category	Severity	Location	Status
Logical Issue	● Minor	ZooSwapMining.sol: 230, 238	✓ Resolved

Description

As the token is a smart contract reference that can be arbitrarily assigned for once-through add() in contract `ZooSwapMining.sol` , its implementation may be unknown to the user and potentialy includes logic to reentrant the sensitive functions such as `withdrawAll()` 、 `withdraw()` .

Recommendation

We advise developers to adopt `nonReentrant` modifier in openzeppelin to sensitive functions `withdrawAll()` and `withdraw()`

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZSM-3 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	ZooSwapMining.sol: 92~94	✓ Resolved

Description

The address of `_zoo` and the address of `_zookeeper` and `_routerAddr` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in constructor of contract `ZooSwapMining.sol` . Violation of this may cause losing ownership of `_zoo` , `_zookeeper` , `_routerAddr` authorization.

Recommendation

Check that the address is not zero by adding following checks in the constructor of contract `ZooSwapMining.sol` .

```
 require(address(_zoo) != address(0));
require(address(_zookeeper) != address(0));
require(_routerAddr != address(0));
```

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

ZSM-4 | Calculation of Reward

Category	Severity	Location	Status
Logical Issue	● Major	ZooSwapMining.sol: 261	✓ Resolved

Description

The below statement:

`user.rewardDebt = user.rewardDebt.add(_amount.mul(pool.accZooPerShare)).div(1e12);`

implies that `rewardDebt` equals to:

`(user.rewardDebt + _amount * pool.accZooPerShare) / 1e12`.

Example:

The below is the result of the first `swap()`:

```
 user.rewardDebt = 0
_amount = 100
pool.accZooPerShare = 1e12
user.rewardDebt.add(_amount.mul(pool.accZooPerShare)).div(1e12) = 100
```

The below is the result of the second `swap()`:

```
 user.rewardDebt = 100
_amount = 100
pool.accZooPerShare = 1e12
user.rewardDebt.add(_amount.mul(pool.accZooPerShare)).div(1e12) = (100+100*1e12)/1e12
```

The result of `rewardDebt` is `(100+100*1e12)/1e12` that is obviously not right.

Recommendation

Consider calculating `rewardDebt` by:

`user.rewardDebt = user.rewardDebt.add(_amount.mul(pool.accZooPerShare).div(1e12));`

Alleviation

The development team heeded our advice and resolved this issue in commit 188e6aabdf56ba77ecfcb479e7f8767ef66be330.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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 exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

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

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings 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

Coding Style findings 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

Inconsistency findings 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

Magic Number findings 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

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

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

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.