auditace.tech

AuditAce

FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

EmojiFinance

DATED : 21 MAR 23'

AUDIT SUMMARY

Project name – EmojiFinance

Date: 21 March, 2023

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: **Passed with High Risk**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	1	3	2	0	2
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1.Manual Review: The code has undergone a line-by-line review by the Ace team.

2.Forked KAVA: we forked EVM co-chain of kava using hardhat, and perfomed all the tests in a realstic environment

3.Slither: The code has undergone static analysis using Slither.

Token Information

Audit ace has performed an security audit of the contracts at this github

https://github.com/EmojiFinance/EmojiProtocol/commits/main

Commit hash:
8b66b73618cf507d5ccddc4e2a918144c139bff5
The repository contains 3 contracts:

- **RewardRelease.sol** (sha-1: f2a6b5685d683114fc4c991d4ed69e7b12ba9e4b)
this contract is used to lock and release rewards of protocol stakers

- **EquiliLpVault.sol** (sha-1 : 56cc4e8fb09fbfa2743bfa286b5ba00bfd7dbb72)
this contract is used to deposit (stake) liquidity pool shares of a curve 2 token pool, the contract generates new pool tokens from generated rewards upon each deposit or withdraw (or manually) and then restakes those tokens to maximize gains.

Token Information

KavaCurveThreePoolVault.sol (sha-1 : 5c4a182ba9600d3659104d914591e875ecbe4cac)
this contract is same as above one, except that its compatible with curve 3 token pools and it interacts directly with pool contract in order to add liquidity and reinvest the new pool shares.

Network: The contracts will be live on Kovan EVM co-chain

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- Return values of low-level calls
- Gasless Send
- Private modifier
- Using block.timestamp
- Multiple Sends
- Re-entrancy
- Using Suicide
- Tautology or contradiction
- Gas Limitand Loops
- Timestamp Dependence
- Address hardcoded
- Revert/require functions
- Exception Disorder
- Use of tx.origin
- Using inline assembly
- Integer overflow/underflow
- Divide before multiply
- Dangerous strict equalities
- Missing Zero Address Validation
- Using SHA3
- Compiler version not fixed
- Using throw

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	1
High-Risk	3
Medium-Risk	2
Low-Risk	0
Gas Optimization / Suggestions	2

INHERITANCE TREE

Current contracts are not implementing or inheriting other contracts

CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
EquiliLpVault	Implementation			
└	<Constructor>	Public ❗️	🛑	NO ❗️
└	setOperator	Public ❗️	🛑	onlyOperator
└	setFeeReceiver	Public ❗️	🛑	onlyOperator
└	setFee	Public ❗️	🛑	onlyOperator
└	setClaimTokens	Public ❗️	🛑	onlyOperator
└	setPaths	Public ❗️	🛑	onlyOperator
└	_setPaths	Internal 🔒	🛑	
└	setRewardRate	Public ❗️	🛑	onlyOperator
└	_updateReward	Internal 🔒	🛑	
└	deposit	Public ❗️	🛑	NO ❗️
└	withdraw	Public ❗️	🛑	NO ❗️
└	withdrawAll	Public ❗️	🛑	NO ❗️
└	getBalance	Public ❗️		NO ❗️
└	pendingReward	Public ❗️		NO ❗️
└	claimReward	Public ❗️	🛑	NO ❗️
└	reinvest	Public ❗️	🛑	NO ❗️
IERC20	Interface			
└	totalSupply	External ❗️		NO ❗️
└	balanceOf	External ❗️		NO ❗️
└	transfer	External ❗️	🛑	NO ❗️
└	allowance	External ❗️		NO ❗️
└	approve	External ❗️	🛑	NO ❗️
└	transferFrom	External ❗️	🛑	NO ❗️
SafeERC20	Library			
└	safeTransfer	Internal 🔒	🛑	
└	safeTransferFrom	Internal 🔒	🛑	
└	safeApprove	Internal 🔒	🛑	
└	safeIncreaseAllowance	Internal 🔒	🛑	
└	safeDecreaseAllowance	Internal 🔒	🛑	
└	safePermit	Internal 🔒	🛑	
└	_callOptionalReturn	Private 🔐	🛑	
IERC20Permit	Interface			
└	permit	External ❗️	🛑	NO ❗️
└	nonces	External ❗️		NO ❗️

CONTRACT ASSESMENT

└	DOMAIN_SEPARATOR	External ❗		NO ❗
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	verifyCallResult	Internal 🔒		
IEquillRouter	Interface			
└	pairFor	External ❗		NO ❗
└	addLiquidity	External ❗	🛑	NO ❗
└	addLiquidityETH	External ❗	💵	NO ❗
└	swapExactTokensForTokensSimple	External ❗	🛑	NO ❗
└	swapExactTokensForTokens	External ❗	🛑	NO ❗
└	swapExactETHForTokens	External ❗	💵	NO ❗
└	swapExactTokensForETH	External ❗	🛑	NO ❗
IEquiliPair	Interface			
└	token0	External ❗		NO ❗
└	token1	External ❗		NO ❗
└	stable	External ❗		NO ❗
IEquiliGauge	Interface			
└	deposit	External ❗	🛑	NO ❗
└	withdraw	External ❗	🛑	NO ❗
└	balanceOf	External ❗	🛑	NO ❗
└	getReward	External ❗	🛑	NO ❗
IRewardRelease	Interface			
└	release	External ❗	🛑	NO ❗
RewardRelease	Implementation			
└	<Constructor>	Public ❗	🛑	NO ❗
└	setOperator	Public ❗	🛑	onlyOperator
└	setVault	Public ❗	🛑	onlyOperator

CONTRACT ASSESMENT

└	release	External ❗	🛑	NO ❗
└	getTotalReward	Public ❗		NO ❗
└	getClaimable	Public ❗		NO ❗
└	claim	External ❗	🛑	NO ❗
KavaCurveThreePoolVault	Implementation			
└	<Constructor>	Public ❗	🛑	NO ❗
└	setOperator	Public ❗	🛑	onlyOperator
└	setFeeReceiver	Public ❗	🛑	onlyOperator
└	setFee	Public ❗	🛑	onlyOperator
└	setRewardRate	Public ❗	🛑	onlyOperator
└	_updateReward	Internal 🔒	🛑	
└	deposit	Public ❗	🛑	NO ❗
└	withdraw	Public ❗	🛑	NO ❗
└	withdrawAll	Public ❗	🛑	NO ❗
└	getBalance	Public ❗		NO ❗
└	pendingReward	Public ❗		NO ❗
└	claimReward	Public ❗	🛑	NO ❗
└	reinvest	Public ❗	🛑	NO ❗
ICurvePool	Interface			
└	add_liquidity	External ❗	🛑	NO ❗
ICurveGauge	Interface			
└	balanceOf	External ❗		NO ❗
└	claimable_reward	External ❗		NO ❗
└	claim_rewards	External ❗	🛑	NO ❗
└	deposit	External ❗	🛑	NO ❗
└	withdraw	External ❗	🛑	NO ❗
└	reward_contract	External ❗		NO ❗

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

FUNCTIONAL TESTING

below tests have been done, the test files (in hardhat) were given to the EmojiFinance team:

- Deployment ✅
- Depositing LP tokens ✅
- Calculation of rewards based on reward rate and shares ✅
- Correct Lock period (2 days) and release of rewards ✅
- Reinves mechanism of both 3 and 2 token pool contracts ✅
- Correct calculation of each depositor reward (failed)
- Withdraw and withdrawAll (failed)

MANUAL TESTING

Logical – Invalid calculation of rewardDebt

Severity: Critical
Function: Deposit
Type: Logical
Line: 164-166
Overview:
user.rewardDebt is divided by 1e12 instead of 10e12, this would disable claims and withdrawals because pending will be almost 1/10 of rewardDebt. This issue can be seen all over the codebase

```
user.pending += accRewardPerShare * user.amount / 10e12 - user.rewardDebt;
user.amount += shareAmount;
user.rewardDebt = user.amount * accRewardPerShare / 1e12;
```

Recommendation:
carefully replace all the 1e12 with 10e12

MANUAL TESTING

Logical – rewardDebt is not calculated correctly upon deposit

Severity: High
Function: Deposit
Type: Logical
Line: 164-166
Overview:
The current implementation of the deposit function updates the user's pending rewards (user.pending) after the user's total deposited amount has been updated. This causes an inconsistency in the calculation of rewardDebt, which represents the user's share of rewards that have already been accounted for. As a result, rewardDebt can be greater than the actual pending rewards, leading to incorrect reward payouts.

```
user.pending += accRewardPerShare * user.amount / 10e12 - user.rewardDebt;
user.amount += shareAmount;
user.rewardDebt = user.amount * accRewardPerShare / 1e12;
```

Recommendation:
To address this issue, we recommend updating the user's rewardDebt and user.pending variables only after the user.amount has been updated. This will ensure that the rewardDebt and pending rewards calculations accurately reflect the user's actual deposited amount and share of rewards.

MANUAL TESTING

Logical – Access control

Severity: High
Function: withdraw
Type: Logical
Line: 188
Overview:
In the present version of the contract, there exists only one operator who has the authority to alter key components in the contract. This operator plays a crucial role and any incorrect modification by this operator can result in the failure of the entire system. For instance, removing the staking contract from valid vaults in the reward contract would disable claims. Similarly, setting up malicious claim tokens and paths can lead to transaction failure during deposit or withdrawal.

Recommendation:
It is recommended to implement a multi-operator scheme to ensure that no single operator has complete control over the contract. This will decrease the risk of the system failure in case of incorrect modifications by the operator. Additionally, the access control system should be reviewed and enhanced to restrict the operator's capabilities to only necessary functions. Regular audits and testing of the contract should also be conducted to identify and rectify any potential vulnerabilities.

MANUAL TESTING

Logical – Lack of escape hatch mechanism (emergency withdraw)

Severity: High
Function: withdraw
Type: Logical
Line: 188
Overview:
An escape hatch mechanism refers to a feature in a software system or a smart contract that allows an authorized entity to override or bypass certain functionalities or restrictions in the system. Essentially, it is a safety mechanism that can be used in an emergency or exceptional circumstances to regain control of the system or to prevent a catastrophic failure. The current version of the contract does not include any emergency withdraw functionality that would enable wallets to bypass or skip regular functionalities of the contract in case of an emergency event or issue.

Recommendation:
It is strongly recommended to implement an emergency withdraw mechanism in the contract to mitigate the risk of potential catastrophic failure.

MANUAL TESTING

Logical – Possible revert on zero token transfer

Severity: Medium
Function: withdraw
Type: Logical
Line: 188
Overview:
At withdraw function, even if withdrawFeeRatio fee amount (0) is sent to feeReceiver, some pools may not allow this kind of transfer in their _transfer function as we see in some implementations of openzeppelin ERC20 contracts where 0 token transfers are not allowed

Recommendation:
check if fee amount is greater than zero or not

MANUAL TESTING

Logical – pendingReward is not calculating rewards correctly

Severity: Medium
Function: Deposit
Type: Logical
Line: 241-247
Overview:
pendingReward function is used to calculate realtime rewards of a staker, however the 10e12 factor is not considered here, which means that if user.amount is not equal to total shares only user.pending will be returned.
Also this function is not considering rewardDebt, which is the paid rewards.

```
function pendingReward(address _user) public view returns (uint256) {
    if (totalShareAmount == 0) {
        return 0;
    }
    UserInfo memory user = userInfo[_user];
    return
        user.pending +
        (user.amount *
            curRewardRate *
            (block.timestamp - lastRewardUpdateTs)) /
        totalShareAmount;
}
```

Recommendation:
Consider multypling the numerator by the 10e12 factor in order to avoid 0 results.
Subtract rewardDebt from the current rewards

MANUAL TESTING

Informational – User rewards are not paid upon withdraw or deposit

Severity: Informational
Function: Deposit
Type: Logical
Line: 143-171
Overview:
The current implementation of the deposit function does not distribute pending rewards to the user when a new deposit is made. Instead, the pending rewards are treated as part of the rewardDebt, which is considered already paid to the user. This approach may lead to inconsistencies in reward payouts and a less intuitive user experience.

Recommendation:
To resolve this issue, we recommend distributing the user's pending rewards upon deposit or withdrawal. This can be achieved by adding a reward distribution function call within the deposit function, such as claimReward() or a similar function, before updating the user's rewardDebt and pending variables. This will ensure that the user's pending rewards are distributed correctly and provide a more intuitive experience.

MANUAL TESTING

Informational – double use of approve function

Severity: **Low (redundant gas)**
Function: Deposit
Type: Informative
Line: ---
Overview:
Within the context of the contract, the approve function is being called twice. The first time is to approve a zero value, and the second time is to approve the desired amount. However, this is unnecessary because the approval of the spender changes to the new amount after the first approval:

```
function approve(address _spender, uint256 _value) public returns (bool)
{
   allowance[msg.sender][_spender] = _value;
   emit Approval(msg.sender, _spender, _value);
   return true;
}
```

Recommendation:
Use the approve function once, unless the target approve function (such as the approve function of LP tokens) has a different implementation that requires two separate approvals.

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace