# Security Assessment

# PKKT

May 25th, 2021

# Table of Contents

# Summary

This report has been prepared for PKKT 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 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 | PKKT |
| Description | An ERC20 token and a farm pool. |
| Platform | Ethereum |
| Language | Solidity |
| Codebase | https://github.com/pokket-finance/pkkt-contract |
| Commits | c2a34555df5a8d88520d7d5f9272116e7e6c3cd9 |

## Audit Summary

| | |
|---|---|
| Delivery Date | May 25, 2021 |
| Audit Methodology | Static Analysis, Manual Review |
| Key Components | |

## Vulnerability Summary

| | |
|---|---|
| Total Issues | 11 |
| Critical | 0 |
| Major | 1 |
| Medium | 0 |
| Minor | 3 |
| Informational | 7 |
| Discussion | 0 |

## Audit Scope

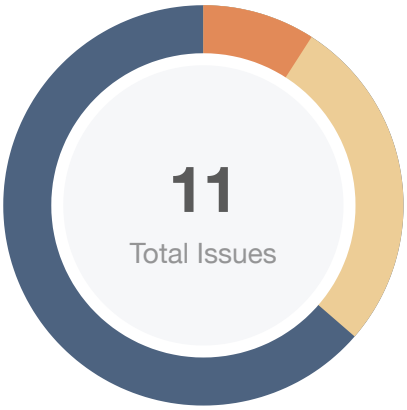
| ID | file | SHA256 Checksum |
| --- | --- | --- |
| PKK | PKKTFarm.sol | 63a34a9ef111711689b0000b6b3d3cb67d296de5aa2595a34613d126b0a6d38a |
| PKT | PKKTToken.sol | 08e27019b9d367420f2945343fd002030647c0a38f59f58e831a3017c2ff68b7 |

It should be noted that the system design includes a number of economic arguments and assumptions. These were explored to the extent that they clarified the intention of the code base, but we did not audit the mechanism design itself.

Additionally, financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. The correctness of the financial model is not in the scope of the audit.

To bridge the trust gap between owner and users, the owner needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The owner has the responsibility to notify users with the following capability of the administrator:
`minter` has the privilege to mint tokens.

# Findings




| | |
|---|---|
| **Critical** | **0** (0.00%) |
| **Major** | **1** (9.09%) |
| **Medium** | **0** (0.00%) |
| **Minor** | **3** (27.27%) |
| **Informational** | **7** (63.64%) |
| **Discussion** | **0** (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| PKK-01 | Comment Typo | Coding Style | Informational | Resolved |
| PKK-02 | `add()` Function Not Restricted | Volatile Code | Major | Resolved |
| PKK-03 | Lack of Pool Validity Checks | Logical Issue | Informational | Resolved |
| PKK-04 | Check Effect Interaction Pattern Violated | Logical Issue | Minor | Resolved |
| PKK-05 | Set `immutable` to Variables | Gas Optimization | Informational | Resolved |
| PKK-06 | Lack of Input Validation | Volatile Code | Minor | Resolved |
| PKK-07 | Proper Usage of `public` and `external` | Gas Optimization | Informational | Resolved |
| **PKT-01** | Owner Capability | **Centralization / Privilege** | **Minor** | **Acknowledged** |
| PKT-02 | Comment Inconsistency | Coding Style | Informational | Resolved |
| PKT-03 | Comparison to A Boolean Constant | Gas Optimization | Informational | Resolved |
| PKT-04 | Set `immutable` to Variables | Gas Optimization | Informational | Resolved |

## PKK-01 | Comment Typo

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | PKKTFarm.sol: 43 | Resolved |

### Description

The comment statement contains a typo in its body, namely `poitns`.

### Recommendation

We advise that the comment text is corrected.

### Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

# PKK-02 | `add()` Function Not Restricted

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Major | PKKTFarm.sol: 76 | Resolved |

## Description

The comment in line L75, mentioned // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.

The total amount of reward `pkktReward` in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code is not reflected in the comment behaviors as there isn't any valid restriction on preventing this issue.

The current implementation is relying on the credibility of the owner to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

## Recommendation

We advise the client to detect whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate.
To achieve this, for example, we advise using mapping of `addresses` -> `booleans`, which can restrict the same address being added twice.

## Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

# PKK-03 | Lack of Pool Validity Checks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | PKKTFarm.sol: 98, 129, 161, 183, 203, 224, 234, 244 | Resolved |

## Description

There's no sanity check to validate if a pool exists.

## Recommendation

We advise the client to adopt following modifier `validatePoolByPid` to functions `set()`, `deposit()`, `withdraw()`, `emergencyWithdraw()`, `pendingPKKT()`, `compoundReward()`, `harvest()` and `updatePool()`.

```
1 modifier validatePoolByPid(uint256 _pid) {
2     require (_pid < poolInfo.length , "Pool does not exist") ;
3     _;
4 }
```

## Alleviation

The team heeded our advice and resolved this issue in commit :
3dce140d4138199ceb6497eaf6059eda3b827990.

# PKK-04 | Check Effect Interaction Pattern Violated

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | PKKTFarm.sol: 227, 192 | Resolved |

## Description

The sequence of external call/transfer and storage manipulation must follow a check effect interaction pattern. For example:

- Function `emergencyWithdraw()`

## Recommendation

We advise the client to adopt `nonReentrant` modifier from openzeppelin library to the function `emergencyWithdraw()` to prevent any reentrancy issue or use the checks-effects-interactions pattern as follows. (LINK)

```
function emergencyWithdraw(uint256 _pid) public{
    ...
    uint amount = user.amount;
    user.amount = 0 ;
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    ...
}

```

## Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

# PKK-05 | Set `immutable` to Variables

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | PKKTFarm.sol: 36, 46 | Resolved |

## Description

The variables `pkkt` and `startBlock` are only changed once in the `constructor()` function.

## Recommendation

We advise the client to set `pkkt` and `startBlock` as `immutable` variables.

## Alleviation

The team heeded our advice and resolved this issue in commit :
3dce140d4138199ceb6497eaf6059eda3b827990.

# PKK-06 | Lack of Input Validation

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | PKKTFarm.sol: 61 | Resolved |

## Description

The assigned values to `pkkt` should be verified as non-zero values to prevent being mistakenly assigned as `address(0)` in the `constructor()` function.

## Recommendation

Check that the addresses are not zero by adding the following check in the `constructor()` function.

```
1  require(address(_pkkt) != address(0), "Zero address");
```

## Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

## PKK-07 | Proper Usage of `public` and `external`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | PKKTFarm.sol: 80, 102, 203, 224, 234 | Resolved |

### Description

`public` functions that are never called by the contract could be declared `external`.

### Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

### Alleviation

The team heeded our advice and resolved this issue in commit :
3dce140d4138199ceb6497eaf6059eda3b827990.

# PKT-01 | Owner Capability

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| **Centralization / Privilege** | **Minor** | PKKTToken.sol: 65 | **Acknowledged** |

## Description

The `minter` has the capability to mint tokens to any address through `mint()`.

## Recommendation

To bridge the trust gap between owner and users, the owner needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The owner has the responsibility to notify users with the following capability of the `minter`:
`minter` has the privilege to mint tokens.

## PKT-02 | Comment Inconsistency

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | PKKTToken.sol: 64 | Resolved |

### Description

The aforementioned comment states that “Must only be called by the owner” which is inconsistent with the codes below.

### Recommendation

We advise that the comment text is corrected.

### Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

# PKT-03 | Comparison to A Boolean Constant

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | PKKTToken.sol: 66 | Resolved |

## Description

Comparison to a boolean constant.

```
1    require(minters[msg.sender] == true , "must have minter role to mint");
```

## Recommendation

Consider removing the comparison to the boolean constant.

## Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

# PKT-04 | Set `immutable` to Variables

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | PKKTToken.sol: 19 | Resolved |

## Description

The variables `_cap` is only changed once in the `constructor()` function.

## Recommendation

We advise the client to set `_cap` as `immutable` variables.

## Alleviation

The team heeded our advice and resolved this issue in commit : 3dce140d4138199ceb6497eaf6059eda3b827990.

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Gas Optimization

Gas Optimization findings 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.

### Logical Issue

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

### Volatile Code

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

### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

# 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.