QuillAudits

Audit Report
November, 2020

KittieFIGHT

Contents

Introduction

This Audit Report mainly focuses on the overall security of KittieFIGHT YieldFarming Smart Contracts. With this report, we have tried to ensure the reliability and correctness of their smart contract by complete and rigorous assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed rigorous testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analysing the complexity of the code in depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analysing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

This audit is based of commit hash `57224c4858c80fe17868ad7ecc8317548dfb3df8` of the GitHub repository - https://github.com/kittiefight/yieldFarming.

The contracts in scope of the audit are YieldFarming.sol, YieldFarmingHelper.sol and, YieldsCalculator.sol.

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity

Issue Categories

Every issue in this report was assigned a severity level from the following:

High severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High
Open	10	0	0
Closed	1	2	1

Manual Audit

High level severity issues

1. Using the `returnTokens` function, the owner of the YieldFarming contract can steal all tokens including all LP tokens, KTY reward tokens, and SDAO reward tokens at any time. It is recommended to add a check in `returnTokens` function that prevents the owner from transferring LP tokens locked by the users.
   Update: The `returnTokens` function has been modified to allow usage only after 60 days of program end time. The users are advised to withdraw their tokens before that.

Medium level severity issues

1. The system heavily depends on the Volcie token that is not in scope for this Audit. The source code for the Volcie token is not present in the repository. If the Volcie token has improper access control checks implemented, the system can be exploited. It is recommended to get the Volcie token audited.
   Update: Volcie token's source code has been added to the repository.

2. The owner can hold LP tokens hostage by changing reward token or withdrawing rewards by using the `returnTokens` function. It is recommended to add a function that burns the Volcie tokens and returns the LP token without trying to give a reward so that the users can at least get their funds back if the Owner's wallet is compromised. This is not as serious of a problem though because the user can re-fund the system with the reward tokens and then burn their volcie token to claim back their rewards.
   Update: The `returnTokens` function has been modified to allow usage only after 60 days of program end time. The users are suggested to withdraw their tokens before that.

Low level severity issues

1. Literals with many digits are difficult to read and review. Constant numbers like `10000000000000000` should be defined as `10 ** 18` to make them more legible.

2. `monthsStartAt` and `programEndAt` are cheaper to calculate on the fly compared to reading them from storage when `programStartAt` is already read. Storage operations like `SLOAD` and `SSTORE` are very expensive in comparison to arithmetic operations. Instead of wasting gas in storing these numbers in storage, they should always be calculated on the fly.

3. In `VolcieToken` struct, `tokenBurnt` and `tokenBurntBy` can be packed together to save one storage slot (20,000 gas). They should be defined one after another so that the compiler can pack them together.

4. `calculated` and `calculated1` state variables are unused and should be removed.

5. `pairCode` is an unneeded abstraction when `pairPoolAddress` can be used directly instead. It is suggested to use `pairPoolAddress` instead of `pairCode` throughout the codebase.

6. On L771 of YieldFarming.sol, `if (_currentMonth < 5)` is not needed since the same condition is checked in the next line (in the for loop). It is recommended to remove this if statement.

7. Reading storage values is quite expensive. At some places like L628 of YieldFarming.sol, the value of `totalNumberOfPairPools` storage variable is read multiple times in a loop. It will be cheaper to cache the value in a local variable and use that in the loop. That being said, this function is a `view` function and is not used by any other smart contract function, therefore, the gas amount being consumed is not important.

8. L486-490 of YieldFarmingHelper, The following code
   ```
   (reserveA,,) = pair.getReserves();
   (,reserveB,) = pair.getReserves();
   ```
   Can be optimized to
   ```
   (reserveA, reserveB,) = pair.getReserves();
   ```

9. In `getMonth` and `getCurrentMonth` functions, the condition used to break the for loop is `i >= 0` where `i` is a uint256 with values ranging from 0 to 2^256 - 1. Therefore, this condition will always be true. It is recommended to convert this condition into `i > 0`

10. There is some duplicated code like the functions `getCurrentMonth` and `getMonth` that can be factored out in a common library contract. In fact, both `YieldFarmingHelper` and `YieldsCalculator` can be made library contracts to make the integration with the `YieldFarming` contract more natural.

11. Test cases are not working and the following error is being returned by Truffle: `Error: Could not find artifacts for Migrations from any sources`. The test cases should be fixed and a test coverage of 100% should be targeted.
    Update: Test cases have been fixed.

Automated Audit

SmartCheck

Smartcheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. Smartcheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false negative rate (FNR). It gave the following result for the relevant contracts:
https://tool.smartdec.net/scan/2ec03db8d2f844168a691d536d07c3c2.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The output generated by Slither can be found at
https://gist.github.com/maxsam4/ad675396c3fe0355e8a2ead2b4f029f5

Disclaimer

This report is not an endorsement or indictment of any particular project or team, and the report does not guarantee the security of any particular project. This report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. This report does not provide any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. I owe no duty to any Third-Party by virtue of publishing these Reports.

The scope of my review is limited to a review of Solidity code and only the Solidity code noted as being within the scope of the review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

This audit does not give any warranties on finding all possible security issues of the given smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, I always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

KittieFIGHT

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com