ChilliSwap

ChilliFarm

Security Assessment

April 8th, 2021

Audited By:
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org
Reviewed By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	ChilliSwap - ChilliFarm
Description	An LP staking implementation based on SushiSwap.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 301dde1383e1707634fce1727d4560c29dc26bea 2. 97c9faa61c28a1937bfaebee869162ddaac975fa

Audit Summary

Delivery Date	April 8th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	March 16th, 2021

Vulnerability Summary

Total Issues	11
Total Critical	0
Total Major	1
Total Medium	1
Total Minor	5
Total Informational	4

Executive Summary

We were tasked with auditing the ChilliSwap codebase, a fork of SushiSwap with a lockup period introduced.

Over the course of the audit we pinpointed 2 severe issues with the system's design as well as code implementation that we advise to be remediated as soon as possible to ensure the system's eligibility for a production deployment.

Additionally, we would like to state that the publicly accessible GitHub repository appears to reveal personally identifying information, including a BitBucket account used in the original development of the codebase as well as Infura API keys which we advise are removed from the repository. This section of the summary will be removed in the final version of the report.

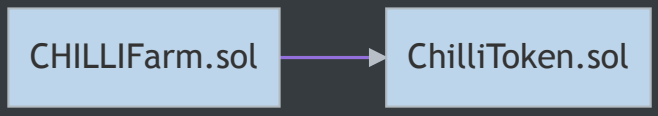
System Analysis

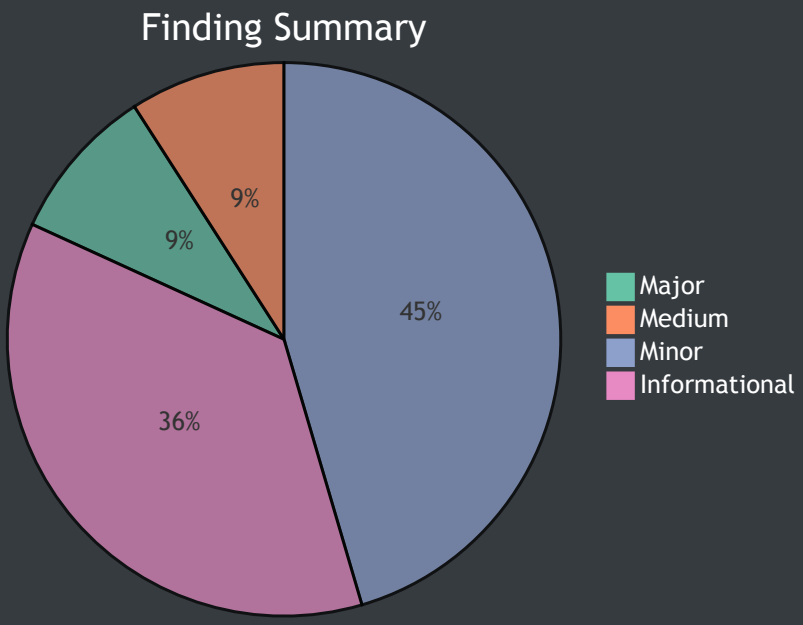
The codebase has introduced certain administrative functions to the original SushiSwap implementation that enable the adjustment of the newly introduced lockup period as well as the replacement of the reward token, in this case Chilli, the latter of which can only effect the rewards paid out and the normal operation of the token and does not inhibit the user's ability to extract their LP tokens from the system via `emergencyWithdraw`.

Files In Scope

ID	Contract	Location
CHI	CHILLIFarm.sol	contracts/CHILLIFarm.sol
CTN	ChilliToken.sol	contracts/ChilliToken.sol

File Dependency Graph





Manual Review Findings

ID	Title	Type	Severity	Resolved
CHI-01M	Permanently Lost Rewards	Logical Issue	Major	✓
CHI-02M	Indiscriminate Entry Timestamp	Logical Issue	Medium	⟳
CHI-03M	Inexplicably Manually-Set Variables	Logical Issue	Minor	✓
CHI-04M	Inefficient Evaluation of Duplicates	Logical Issue	Minor	✓
CHI-05M	Introduction of Race Condition	Logical Issue	Minor	⟳
CHI-06M	Redundant Statements	Dead Code	Informational	✓

Static Analysis Findings

ID	Title	Type	Severity	Resolved
CHI-01S	Inapplicacy of Checks-Effects-Interactions Pattern	Logical Issue	Minor	✓
CHI-02S	Lack of Zero-Address Check	Logical Issue	Minor	✓
CHI-03S	Redundant `constructor` Visibility	Coding Style	Informational	✓
CTN-01S	Redundant Statements	Dead Code	Informational	✓
CTN-02S	Long Numeric Literals	Coding Style	Informational	✓

CHI-01M: Permanently Lost Rewards

Type	Severity	Location
Logical Issue	Major	CHILLIFarm.sol L238, L273, L274

Description:

The system appears to contain a miswritten implementation of a system whereby rewards are accumulated in a `pendingChillies` mapping before being paid out by their own dedicated function. However, in its current state, pending rewards are stored in the `pendingChillies` mapping solely when a new `deposit` occurs and they can never be redeemed as the `claim` function they were intended to be utilized in is commented out. Thus, rewards are overwritten on each `deposit`.

Recommendation:

We advise the system is adjusted to properly pay out pending rewards, as the accumulation of rewards in the `pendingChillies` is done so to prevent early redemption of rewards before the lock up period.

Alleviation:

The system is still not performing as expected. The commented out line of L274 (now L266) was uncommented, however, it accumulates pending chillies to the mapping after paying them out which is not logical. Additionally, the `pendingChillies` mapping is never utilized in any payout and thus all chillies set to it i.e. via the `deposit` function will be permanently lost.

This issue was completely fixed in the latest iteration of the codebase, whereby accumulated rewards are properly zeroed out and added to when necessary. We should add that the Checks Effects Interactions pattern is not applied in the `withdraw` function of the contract whereby the `pendingChillies` entry is utilized in the transfer before it is zeroed out that could lead to a re-entrancy, however, the Chilli implementation in its current state does not permit a re-entrancy in a transfer and thus the Checks Effects Interactions can optionally be

applied but does not pose a security threat to the system.

CHI-02M: Indiscriminate Entry Timestamp

Type	Severity	Location
Logical Issue	Medium	CHILLIFarm.sol L243, L260-L263

Description:

The `entryTimestamp` member of the `UserInfo` structure is meant to introduce a new feature to the SushiSwap code the project is based on whereby withdrawals are prohibited until a certain lock period has passed. In this implementation, the `entryTimestamp` variable is updated on each `deposit` regardless of the pool token being updated which can lead to misleading behaviour for the users.

Recommendation:

We advise the lock period to be enforced on a per-pool basis rather than on a per-account basis to ensure users can freely deposit into other pools as they are introduced without affecting their previous deposits.

Alleviation:

The ChilliSwap team stated that they do not intend to introduce a timestamp on a per-pool basis as that would require a different storage variable for each slot which would require a big overhaul of the system.

CHI-03M: Inexplicably Manually-Set Variables

Type	Severity	Location
Logical Issue	Minor	CHILLIFarm.sol L46, L69, L71, L327-L340

Description:

The three linked variables denote core components of the system and are entirely controlled by the developers of the project and can arbitrarily be changed.

Recommendation:

While the `lockPeriod` being adjusted is understandable, being able to swap the underlying `chilli` token is ill practice and we advise the setter to be removed and the token to be assigned during the `constructor` optimizably so to an `immutable` variable.

In regards to the `feesAccumulated` being adjustable, there is no clear purpose for this functionality as the variable is not utilized within the codebase and acts as a simple member of the contract that can be set at will and is not guaranteed to reflect reality.

Alleviation:

A new `require` check was introduced to the codebase that is meant to permit the setting of the `chilli` address only once, however, it achieves the opposite whereby the `chilli` address can never be set as the condition within the introduced `require` check should be inverted.

The `require` case was inverted in the latest version of the codebase thereby alleviating this issue.

CHI-04M: Inefficient Evaluation of Duplicates

Type	Severity	Location
Logical Issue	Minor	CHILLIFarm.sol L108-L113

Description:

The `checkPoolDuplicate` function evaluates whether a designated `_lpToken` has already been added to the ChilliSwap system to prevent it from being added twice. The implementation can become prohibitively expensive as more pools are added to the system because it iteratively loops over all entries and evaluates a `require` check within the loop.

Recommendation:

We advise a `mapping` to be utilized instead which links an `_lpToken` with a `bool` variable and is set to `true` whenever a new `_lpToken` is added to the system. This will result in significantly less gas to be utilized for the `checkPoolDuplicate` evaluation and permits the system to scale without an issue.

Alleviation:

A `mapping` was introduced to make the process more efficient, however, in doing so a new vulnerability was introduced whereby new pools cannot be created as the `checkPoolDuplicate` function evaluates whether `supportedPools[_lpToken]` is equal to `true` instead of evaluating that it is equal to `false` .

This issue was corrected in the latest commit of the project thereby alleviating this exhibit in full.

CHI-05M: Introduction of Race Condition

Type	Severity	Location
Logical Issue	Minor	CHILLIFarm.sol L224

Description:

The system of ChilliSwap does not perpetually mint tokens and instead relies on a predefined supply of tokens that is allocated during the native token's creation. This indirectly introduces a race condition whereby users that have accumulated significant rewards towards the end of the remaining balance of the contract will conduct a "gas-war" to try and include their transaction in front of others to acquire the last rewards and cause the other transactions to fail.

Recommendation:

We advise this trait of the system to be thoroughly evaluated as it can lead to misbehavior that user's may not expect since the widely known functionality of such LP pools is that they can be idly farmed instead of actively.

Alleviation:

The ChilliSwap - ChilliFarm development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

CHI-06M: Redundant Statements

Type	Severity	Location
Dead Code	Informational	CHILLIFarm.sol L33-L37, L62-L63, L281-L298

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

The redundant statements of the codebase as well as the commented out function were properly removed.

CHI-01S: Inapplicacy of Checks-Effects-Interactions Pattern

Type	Severity	Location
Logical Issue	Minor	CHILLIFarm.sol L302-L309

Description:

The `emergencyWithdraw` function does not conform to the Checks-Effects-Interactions pattern and utilizes input variables in an external call that are zeroed out after its conclusion.

Recommendation:

While the risk here is practically inexistent as the `lpToken` implementation of the pool does not contain logic that will inform the recipient of the transfer, it is still considered best practice to apply the Checks-Effects-Interactions on the codebase and we advise it to be done so by caching the `user.amount` and zeroing it from `storage` before the external `safeTransfer` call.

Alleviation:

The Checks-Effects-Interactions pattern was successfully applied by storing the `user.amount` in an intermediate variable, zeroing it out and then performing the external call.

CHI-02S: Lack of Zero-Address Check

Type	Severity	Location
Logical Issue	Minor	CHILLIFarm.sol L82, L88, L322-L325, L327-L330

Description:

The `_devaddr` constructor variable is not restricted to not be the zero-address, potentially leading to a misconfigured farm state.

Recommendation:

We advise that a zero address check is imposed in the `constructor` via a `require` check ensuring that `_devaddr` is different than the zero address.

Alleviation:

A zero-address check was introduced in the `constructor` of the contract ensuring that the `_devaddr` is not equal to the zero address.

CHI-03S: Redundant `constructor` Visibility

Type	Severity	Location
Coding Style	Informational	CHILLIFarm.sol L87

Description:

The visibility specifier for a `constructor` is ignored in the version of Solidity utilized by the project, `0.7.5`.

Recommendation:

We advise the visibility specifier to be removed from the codebase.

Alleviation:

The `constructor` 's visibility specifier was properly removed from the codebase.

CTN-01S: Redundant Statements

Type	Severity	Location
Dead Code	Informational	ChilliToken.sol L8

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

The `farmingContract` variable was safely omitted from the codebase.

CTN-02S: Long Numeric Literals

Type	Severity	Location
Coding Style	Informational	ChilliToken.sol L19-L25

Description:

The linked statements contain numeric literals with more than 4 digits that can be hard to read, especially when they are similar like `2400000` and `24000000`.

Recommendation:

We advise the inclusion of a separator, `_`, between the thousand denominators to increase the legibility of the codebase. The `_` character is ignored in numeric literals and can be safely used in the implementation. Additionally, we advise the `10 ** 18` variable to be stored as a contract-level `constant` further increasing legibility and allowing the contract to be maintained easily i.e. adjusting its decimals.

Alleviation:

The numeric literals were properly formatted to be more readable and a `multiplier` contract-level `constant` was declared that contains the token unit offset. We should note that it is more advisable to change the `multiplier` variable name to `MULTIPLIER` as per the official Solidity style guide.

Appendix

Finding Categories

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.

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.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.