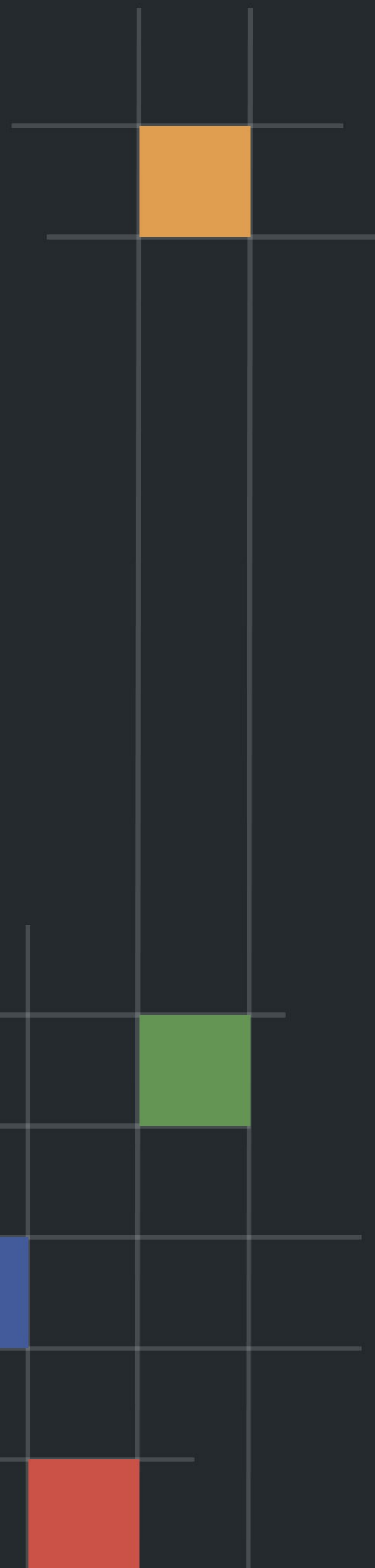




Wasabi Finance

Security Assessment

Mar 19th 2021



[Summary](#)

[Overview](#)

[Project Summary](#)

[Engagement Summary](#)

[Finding Summary](#)

[Findings](#)

[CTK-WASABI-1 | Proper Usage of public and external](#)

[CTK-WASABI-2 | Checks Effects Interaction Pattern Not Used](#)

[CTK-WASABI-3 | Checks Effects Interaction Pattern Not Used](#)

[CTK-WASABI-4 | Missing Zero Address Validation](#)

[CTK-WASABI-5 | Function Return Value Ignored](#)

[CTK-WASABI-6 | Missing Emit Events](#)

[CTK-WASABI-7 | Privileged Ownerships on MasterChef](#)

[CTK-WASABI-8 | Privileged Ownerships on WasabiToken](#)

[CTK-WASABI-9 | Multiplication on the result of a division](#)

[CTK-WASABI-10 | add\(\) Function Not Restricted](#)

[CTK-WASABI-11 | Gas Optimization](#)

[Appendix | Finding Categories](#)

[Disclaimer](#)

[About CertiK](#)

Summary

This report has been prepared for Wasabi Finance smart contracts, MasterChef, WasabiToken, ContributorsVault, TeamsVault and libs to discover issues and vulnerabilities in the source code as well as any 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 security experts.

The security assessment resulted in 10 findings that ranged from Major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices.

We suggest below recommendations that could better serve the project from the security perspective:

1. Enhance general coding practices for better structures of source codes;
2. Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
3. Provide more comments per each function for readability, especially contracts are verified in public;
4. Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Name	Wasabi Finance
Codebase	https://github.com/wasabi-swap-team/wasabi-swap-farm/tree/main/contracts
Commit Hash	e5e96bff14d619f7bf3855872c542c35257338b3

Engagement Summary

Delivery Date	Mar 19th, 2021
Methodology	Static analysis and manual review
Contracts in Scope	5
Contract - Token	WasabiToken
Contract - MasterChef	MasterChef
Contract - ContributorsVault	ContributorsVault
Contract - TeamsVault	TeamsVault
Contract - StakedWasabi	StakedWasabi

Finding Summary

Total	11
Major	2

Medium	0
Minor	4
Informational	5

Findings

ID	Title	Severity	Response
CTK-WASABI-1	Proper Usage of public and external	Informational	Pending
CTK-WASABI-2	Checks Effects Interaction Pattern Not Used	Minor	Pending
CTK-WASABI-3	Checks Effects Interaction Pattern Not Used	Major	Pending
CTK-WASABI-4	Missing zero address validation	Minor	Pending
CTK-WASABI-5	Function Return Value Ignored	Informational	Pending
CTK-WASABI-6	Missing Emit Events	Informational	Pending
CTK-WASABI-7	Privileged Ownerships on MasterChef	Informational	Pending
CTK-WASABI-8	Privileged Ownerships on WasabiToken	Informational	Pending
CTK-WASABI-9	Multiplication on the result of a division	Informational	Pending
CTK-WASABI-10	add() Function Not Restricted	Major	Pending
CTK-WASABI-11	Gas Optimization	Informational	Pending

CTK-WASABI-1 | Proper Usage of public and external

Type	Severity	Location
Gas Optimization	Informational	MasterChef

Description

`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays `external` functions are more efficient than `public` functions.

MasterChef:

```
updateWasabiPerBlock(), updateRewardPercentage(), setBootstrappingValid(),  
migrate()
```

Recommendation

Consider using the `external` attribute for functions never called from the contract.

CTK-WASABI-2 | Checks Effects Interaction Pattern Not Used

Type	Severity	Location
Logic issue	Minor	SmartChef.sol: L275, L318

Description

In function `deposit()` and `enterStaking()`, `lpToken` is pointing to a smart contract that is implemented based on an ERC20 interface. This smart contract can only be passed into the function `add()` by `owner` as one of the parameters while the implementation of `lpToken` is unknown statically, even if it strictly followed the ERC20 interface.

Due to the unknown implementation of contract `lpToken`, the implementation of function `safeTransferFrom()` in L289 is also unknown and may have a malicious logical implementation that calls back to the function `deposit()`, which can lead to another invocation of `deposit()` without updating `user.rewardDebt` in L283. This will incorrectly refund pending rewards multiple times to the user.

Recommendation

We advise developers to update the value of `user.rewardDebt` before `pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);` to follow the [Checks-Effects-Interactions Pattern](#).

```
function deposit(uint256 _pid, uint256 _amount) public {
    ...
    if (user.amount > 0) {
        uint256 pending = user.amount.mul(pool.accWasabiPerShare).div(1e12)
        .sub(user.rewardDebt);
        user.rewardDebt = user.amount.mul(pool.accWasabiPerShare).div(1e12); //update
        if (pending > 0) {
            safeWasabiTransfer(msg.sender, pending);
        }
    }
    if (_amount > 0) {
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);
    }
    user.rewardDebt = user.amount.mul(pool.accWasabiPerShare).div(1e12);
}
```

CTK-WASABI-3 | Checks Effects Interaction Pattern Not Used

Type	Severity	Location
Logic issue	Major	SmartChef.sol: L359

Description

In function `emergencyWithdraw()`, `pool.lpToken` is pointing to a smart contract that is implemented based on an ERC20 interface. This smart contract can only be passed into the function `add()` by `owner` as one of the parameters while the implementation of `lpToken` is unknown statically, even if it strictly followed the ERC20 interface.

Due to the unknown implementation of contract `lpToken`, the implementation of function `safeTransfer()` in L362 is also unknown and may have a malicious logical implementation that calls back to the function `emergencyWithdraw()`, which can lead to another invocation of `emergencyWithdraw()` without updating `user.amount` in L364. This is dangerous to the `user.amount` and will incorrectly withdraw multiple times to the `msg.sender`.

Recommendation

We advise developers to update the value of `user.amount` before `pool.lpToken.safeTransfer(address(msg.sender), user.amount);` to follow the [Checks-Effects-Interactions Pattern](#).

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 withdraw_amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
    pool.lpToken.safeTransfer(address(msg.sender), withdraw_amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
}
```

CTK-WASABI-4 | Missing Zero Address Validation

Type	Severity	Location
Volatile Code	Minor	MasterChef: L373, L377 ContributorsVault: L118 L178

Description

The assigned value to `vault`, `dev` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in `updateDevAddress()` function and `updateVaultAddress()`. Violation of this may cause losing ownership of `vault` and `dev`.

Recommendation

Check that the address is not zero by adding checks in function `updateVaultAddress()` and `updateDevAddress()`. Please ignore if the team inclines to leverage the same function in a way to renounce the fee collections (mimic the token burn in a way)

CTK-WASABI-5 | Function Return Value Ignored

Type	Severity	Location
Volatile Code	Informational	StakedWasabi: L40, L42

Description

The return values of `wasabi.transfer(_to, wasabiBal), wasabi.transfer(_to, _amount);` are ignored in function `safeWasabiTransfer()`.

Recommendation

We advise developers to handle the return value of `wasabi.transfer()` to check if the transfer is executed without any error.

CTK-WASABI-6 | Missing Emit Events

Type	Severity	Location
Volatile Code	Informational	MasterChef.sol

Description

Functions that affect the status of sensitive variables should be able to emit events as notifications to customers.

MasterChef:

```
migrate(),updateRewardPercentage(),updateWasabiPerBlock()
```

Recommendation

Consider adding events for sensitive actions, and emit them in the function like below.

```
event Migrate(address indexed user, uint256 indexed _pid);
...
function migrate(uint256 _pid) public{
...
    emit Migrate(msg.sender, _pid);
}
```

CTK-WASABI-7 | Privileged Ownerships on MasterChef

Type	Severity	Location
Business Model	Informational	MasterChef: L127, L131, L172

Description

The owner of MasterChef has permission to update the parameters on rewards without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

CTK-WASABI-8 | Privileged Ownerships on WasabiToken

Type	Severity	Location
Business Model	Informational	WasabiToken, StakedWasabi

Description

WasabiToken and StakedWasabi are standard ERC20 implementations that contain the mint functionality with ownership controls, which means whoever obtained access to the owner account would be able to tamper with the integrity of the token economics.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations. Specifically for this scenario, we assume the owner will be transferred to the vault (MasterChef) on top of the token. We recommend that the team maintains a high level of transparency on such a transaction taking place.

CTK-WASABI-9 | Multiplication on the result of a division

Type	Severity	Location
Volatile Code	Minor	ContributorsVault L214, L241, L253

Description

L213-L214:

```
uint256 lpWasabiPerBlock = wasabiPerBlock.mul(lpRewardPercentage).div(100);  
uint256 wasabiReward = multiplier.mul(lpWasabiPerBlock).mul(allocPoint)  
.div(totalAllocPoint);
```

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

Consider ordering multiplication before division.

CTK-WASABI-10 | add() Function Not Restricted

Type	Severity	Location
Volatile Code	Major	MasterChef: L157

Description

The comment in L157, 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 `lpWasabiReward` 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 trust 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

Using mapping of `addresses -> booleans`, which can restrict the same address being added twice.

CTK-WASABI-11 | Gas Optimization

Type	Severity	Location
Business Model	Informational	ContributorsVault L67-75

Description

Refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction in the total gas cost of a transaction.

Recommendation

We recommend using following implementation to save gas:

```
for (uint256 i = 0; i < beneficiaries_.length; i++) {  
    require( beneficiaries_[i] != address(0) && amounts_[i] > 0 ,  
    "Beneficiary address or allocate cannot be zero.")  
    ... ..  
}
```

Appendix | Finding Categories

Gas Optimization

Refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction in the total gas cost of a transaction.

Mathematical Operations

Refer to exhibits that relate to mishandling of math formulas, such as overflows, incorrect operations, etc.

Logical Issue

Refer to exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

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

Volatile Code

Refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

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

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

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

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

Refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

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

Business Model

Refer to contract or function logics that are debatable or not clearly implemented according to the design intentions.

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.

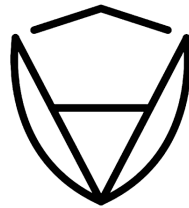
This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security 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 CertiK

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.



CERTiK
Provable Trust For All