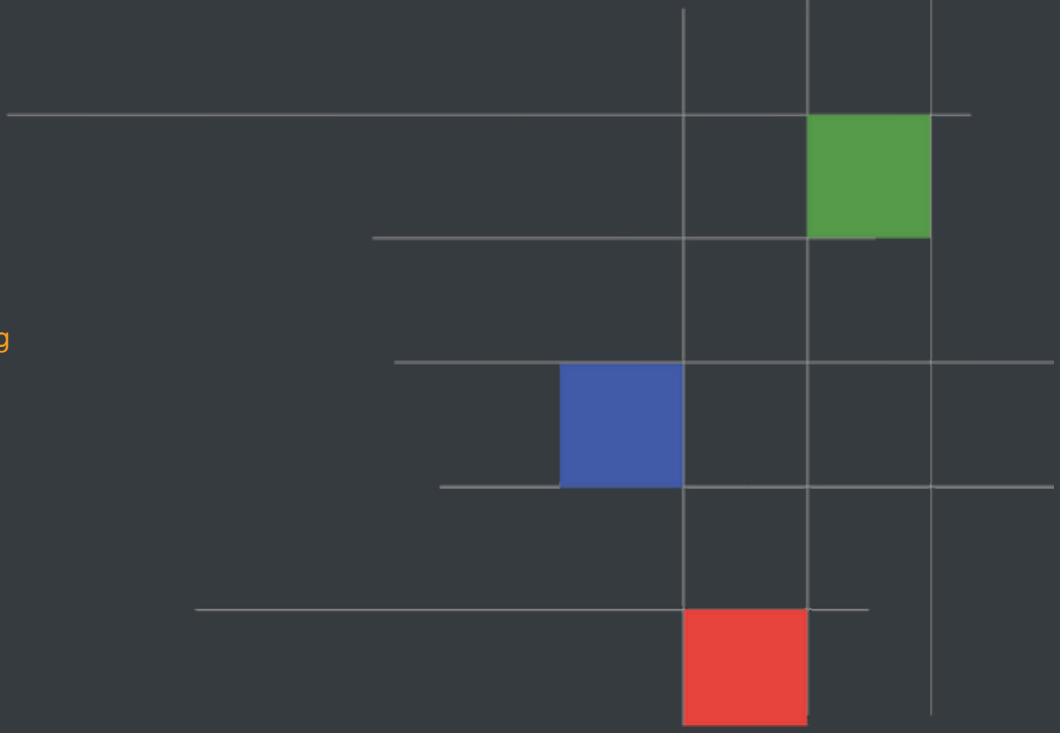
Warden

WardenSwap Farm

Security Assessment

May 7th, 2021



Audited By:
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org
Reviewed By:
Alex Papageorgiou @ CertiK
alex.papageorgiou@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	Warden - WardenSwap Farm
Description	WardenSwap yield farming codebase
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. cd769cd3dada5e37f5a89c962d60e134aa0c6f99 2. 82548f2ed184f4be4acce2c951eced0c98510642

Audit Summary

Delivery Date	May 7th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	April 23rd, 2021 - May 7th, 2021

Vulnerability Summary

Total Issues	3
Total Critical	0
Total Major	1
Total Medium	0
Total Minor	2
Total Informational	0

Executive Summary

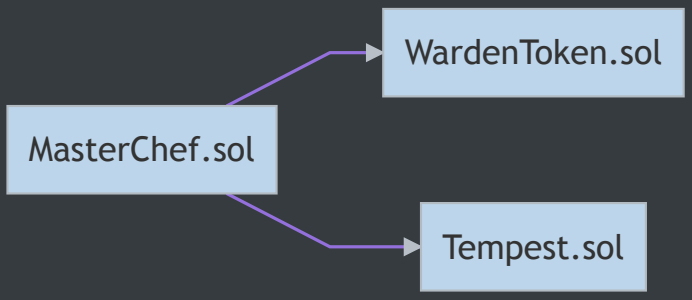
The report represents the results of CertiK's engagement with Warden on the implementation of their yield farming smart contracts.

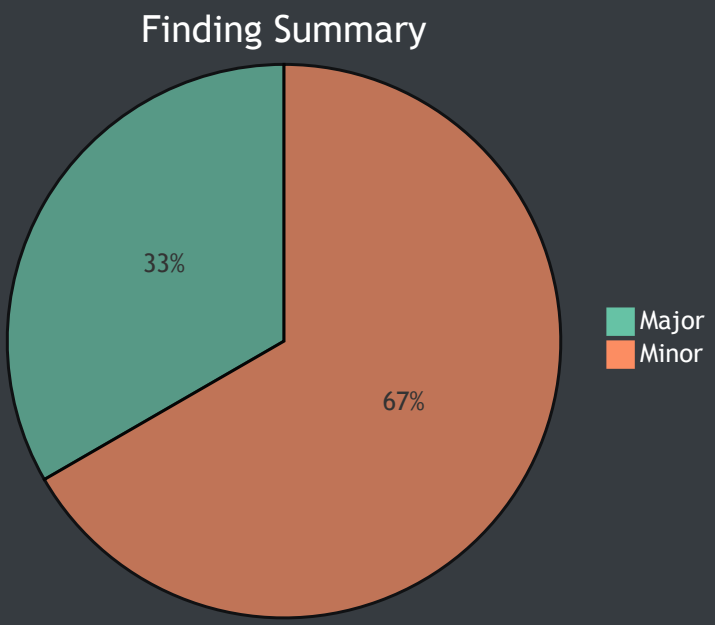
No notable vulnerabilities were identified in the codebase and it makes use of the latest security principles and style guidelines. It should be noted that the codebase is based on a known project, hence inheriting the original's vulnerabilities.

Files In Scope

ID	Contract	Location
MCF	MasterChef.sol	contracts/MasterChef.sol
TEM	Tempest.sol	contracts/Tempest.sol
TIM	Timelock.sol	contracts/Timelock.sol
WTN	WardenToken.sol	contracts/WardenToken.sol
MUL	Multicall.sol	contracts/libs/Multicall.sol

File Dependency Graph





Manual Review Findings

ID	Title	Type	Severity	Resolved
MCF-01	Ambiguous Functionality	Logical Issue	Major	✓
TEM-01	Possibility of Replay Attack in `Permit`	Logical Issue	Minor	✓
WTN-01	Possibility of Replay Attack in `Permit`	Logical Issue	Minor	✓

MCF-01: Ambiguous Functionality

Type	Severity	Location
Logical Issue	Major	MasterChef.sol L308-L318

Description:

The `emergencyWithdraw()` function should not allow withdrawing from the first pool, as it will not burn the Tempest tokens.

Recommendation:

We advise to add a `require` statement, ensuring that a user cannot withdraw from the first pool via the linked function.

Alleviation:

The development team opted to consider our references and added a `require` statement, as proposed, ensuring that a user cannot withdraw from the first pool via the `emergencyWithdraw()` function.

TEM-01: Possibility of Replay Attack in `Permit`

Type	Severity	Location
Logical Issue	Minor	Tempest.sol L37-L45

Description:

The `permit` function on `L292` performs the operation of deriving signer address from the signature values of `v`, `r` and `s`. The state varible `DOMAIN_SEPARATOR` that is used to calculate hash has a value of `chainid` that is derived only once in initialize function, which does not change after contract deployment. The issue arises in the event of fork when the cross-chain replay attacks can be executed.
The attack scenario can be thought of as if a fork of Ethereum happens and two different networks have id of for example `1` and `9`. The `chainid` coded in `DOMAIN_SEPARATOR` will be the same on contracts residing in both of the forks. If the `chainid` `1` is stored in the contract then the `permit` transaction signed for `chainid` `1` will be executable on both of the forks.

Recommendation:

We advise to construct the `DOMAIN_SEPRATOR` hash inside the `permit()` function so the current `chainid` could be fetched and only the transactions signed for current network could succeed.

Alleviation:

The development team opted to consider our references and constructed the `DOMAIN_SEPRATOR` hash inside the `permit()` function.

WTN-01: Possibility of Replay Attack in `Permit`

Type	Severity	Location
Logical Issue	Minor	WardenToken.sol L62-L70

Description:

The `permit` function on `L267` performs the operation of deriving signer address from the signature values of `v`, `r` and `s`. The state varible `DOMAIN_SEPARATOR` that is used to calculate hash has a value of `chainid` that is derived only once in initialize function, which does not change after contract deployment. The issue arises in the event of fork when the cross-chain replay attacks can be executed.
The attack scenario can be thought of as if a fork of Ethereum happens and two different networks have id of for example `1` and `9`. The `chainid` coded in `DOMAIN_SEPARATOR` will be the same on contracts residing in both of the forks. If the `chainid` `1` is stored in the contract then the `permit` transaction signed for `chainid` `1` will be executable on both of the forks.

Recommendation:

We advise to construct the `DOMAIN_SEPRATOR` hash inside the `permit()` function so the current `chainid` could be fetched and only the transactions signed for current network could succeed.

Alleviation:

The development team opted to consider our references and constructed the `DOMAIN_SEPRATOR` hash inside the `permit()` function.

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.