CertiK Round II Audit Report for Onther

Contents

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 Verification Services Agreement between CertiK and Onther (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK’s mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance’s BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report acts as the second round report that identifies findings in the codebase of the following repositories and corresponding commit hashes:

- https://github.com/Onther-Tech/presale-contracts/tree/8fc3514285f610e531eb4159cd32552abe9169e7

Testing Summary

SECURITY LEVEL

Smart Contract Audit
This report has been prepared as a product of the Smart Contract Audit request by Onther.
This audit was conducted to discover issues and vulnerabilities in the source code of the Onther Smart Contracts.

TYPE	Smart Contract
SOURCE CODE	https://github.com/Onther-Tech/presale-contracts/tree/8fc3514285f610e531eb4159cd32552abe9169e7
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 02, 2020
DELIVERY DATE	Aug 11, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the Onther team to audit the design and implementation of their smart contracts and its compliance with the security standards it is meant to implement.

The audited source code link is:

- https://github.com/Onther-Tech/presale-contracts/tree/8fc3514285f610e531eb4159cd32552abe9169e7

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

Documentation

The sources of truth regarding the operation of the contracts in scope were minimal although the smart contracts fulfilled a simple use case we were able to fully assimilate. To help aid our understanding of each contract's functionality we referred to the thorough in-line comments and naming conventions.

Summary

The codebase of the project is a typical ERC20 implementation with vesting mechanisms, token sale implementations as well as a vault system.

Although **certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies, **any minor (or above) flaws** that were identified, **should be remediated as soon as possible to ensure the security of the contracts** of Onther's team.

The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and as such its functions **can be deemed to be of high security and quality, however the custom functionality built on top of it possessed flaws** we identified.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.
Most of the findings we pointed out were swiftly dealt with by the Onther team and as such, we consider the codebase of the project of great quality and security.

Findings

Burner.sol (BNR)

A file presumably implementing burning functions for a token.

BNR-01: No Implementation

Description

The contract has unnecessary imports and otherwise conducts no logic within its body. As this contract is meant to be deployed to signify an Ethereum `address` as the "burn" address, it makes more sense to instead utilize a `constant` declared so. Additionally, the linked `using` notation for SafeMath is redundant as no math is conducted within the contract.

Remediation

The contract properly consists of an empty contract body, possessing no redundant imports.

Swapper.sol (SWP)

A token swapper mechanism.

SWP-01A , SWP-01B: Variable Visibility Specifiers

Description

The linked variables do not have a visibility specifier set. We advise that an explicit visibility specifier is set for these variables to aid in the legibility of the codebase.

Remediation

Visibility specifiers were properly set for the linked variables.

SWP-02: Inexistent Re-invocation Guard

Description

The linked `setStart` function can be arbitrarily invoked in the future overriding the previous `startTimestamp` variable at any time leading to unaccounted for consequences. We advise that a `require` check is imposed that ensures `startTimestamp` is zero.

Remediation

A sanity check was imposed whereby `startTimestamp` is ensured to be equal to zero before being assigned to a value.

SWP-03: Comparison Inefficiency

Description

As the linked comparisons conduct so with unsigned integers, it is possible to restrict their greater-than comparisons with zero to inequality comparisons which are also more gas-efficient.

Remediation

The comparisons were properly adjusted to their optimized inequality counterparts.

SWP-04A , SWP-04B: Redundant Transfer Roundtrips

Description

The linked code statements transfer the funds out of a user's account to the contract via a `transferFrom` call and subsequently transfers them to another address via a `transfer` call.

We advise that a direct `transferFrom` call is utilized from the user's address to the destination address to greatly reduce the gas cost of the function.

Remediation

The Onther team responded by stating that the token they are meant to interact with would not allow a direct `transfer` call between the two addresses and as such, the round-trip is deemed necessary.

SWP-05: Variable Naming

Description

The linked variable is meant to represent the burner address yet is named `bernerAddress`. We advise that this typo is fixed.

Remediation

The burner address setter was removed form the codebase, presumably because the "reserved" address `0x00...00` will be utilized, thus nullifying this exhibit.

TONVault.sol (TVL)

This contract appears to be an incomplete implementation of a token vault that is locked up for a specific period of time.

TVL-01: Unused Variable

Description

The variable `withdrawableTime` remains unused throughout the contract yet indicates further development is expected on the contract.

Remediation

The unused variable was omitted from the codebase of the contract.

VestingSwapper.sol (VSW)

A vested token swapper mechanism.

VSW-01A , VSW-01B , VSW-01C: Variable Visibility Specifiers

Description

The linked variables do not have a visibility specifier set. We advise that an explicit visibility specifier is set for these variables to aid in the legibility of the codebase.

Remediation

Visibility specifiers were properly set for the linked variables.

VSW-02: Inefficient Struct

Description

The current implementation utilizes 9 full word slots, a.k.a. 32-byte / 256-bit slots, whereas it can at minimum be reduced to 8 full word slots by tight packing the `bool` variable with a `uint` that is of size `248` instead of `256`.

Additionally, the other variables should be evaluated and correspondingly packed as f.e. a unix timestamp has at most 64-bits of precision even on contemporary systems.

Remediation

The variable sizes were reduced greatly optimizing the gas cost involved with interacting with the `VestingInfo` struct. A simple note we would like to make is that the current structure does contain empty bits and would be better to not leave trailing bits by adjusting the variable sizes `ratio` and / or `start` to have the three first variables of the struct occupy a full word i.e. 256-bits.

VSW-03A , VSW-03B , VSW-03C , VSW-03D: Comparison Inefficiency

Description

As the linked comparisons conduct so with unsigned integers, it is possible to restrict their greater-than comparisons with zero to inequality comparisons which are also more gas-efficient.

Remediation

The comparisons were properly adjusted to their optimized inequality counterparts.

VSW-04: Invalid `require` Reason

Description

The `require` reason and consequent comment should be properly revised as they indicate leftover test code.

Remediation

The `require` reason was properly set yet a trailing comment remains which should be omitted.

VSW-05A , VSW-05B: Inexistent `require` Reasons

Description

Proper `require` reasons should be supplied to the linked lines as it is a standard coding practice to do so.

Remediation

Error strings for the corresponding `require` statements were supplied.

VSW-06: Inexistent Re-invocation Guard

Description

The linked `setStart` function can be arbitrarily invoked in the future overriding the previous `startTimestamp` variable at any time leading to unaccounted for consequences. We advise that a `require` check is imposed that ensures `startTimestamp` is zero.

Remediation

A sanity check was imposed whereby `startTimestamp` is ensured to be equal to zero before being assigned to a value.

VSW-07: Instantiate vs. Assign Pattern

Description

The current way the `VestingInfo` struct is initialized is very inefficient as it performs heavy storage writes on each line assignment as well as mapping lookups. We advise that an instantiation of a `VestingInfo` struct in-memory is conducted that is subsequently assigned to the `storage` slot of `vestingInfo[vestingToken]`.

Remediation

An in-memory struct declaration called `info` was set and subsequently assigned to the storage slot of `vestingInfo[vestingToken]`, applying our exhibit in full.

VestingToken.sol (VST)

A contract inheriting the `MiniMeToken.sol` and implementing a vesting token similarly to `TokenVesting.sol`.

VST-01: Redundant Getter Functions

Description

These functions can be safely omitted by removing the underscore prefix from the variables they are meant to return and declaring the said variables as `public`.

Remediation

Our recommendation was not applied on the codebase as it is an optimization and the Onther team decided not to apply it.

VST-02: Inexistent Sanity Check

Description

The result of `start.add(cliffDuration)` should logically be less than the result of `start.add(duration)` yet this remains unchecked in the codebase. We advise that this is imposed via a `require` check in the linked function.

Remediation

The logical comparison is checked during initialization, however we believe this should be a strict less-than comparison (`<`) instead of a less-than-or-equal (`<=`) comparison.

VST-03A , VST-03B: Comparison Inefficiency

Description

As the linked comparisons conduct so with unsigned integers, it is possible to restrict their greater-than comparisons with zero to inequality comparisons which are also more gas-efficient.

Remediation

Our recommendation was not applied on the codebase as it is an optimization and the Onther team decided not to apply it.

VestingTokenStep.sol (VSS)

A vesting token implementation that also includes a stepping mechanism whereby the unlock stages of the vesting are stepped instead of linear.

VSS-01: Variable Visibility Specifiers

Description

The linked variables do not have a visibility specifier set. We advise that an explicit visibility specifier is set for these variables to aid in the legibility of the codebase.

Remediation

Visibility specifiers were properly set for the linked variables.

VSS-02: Redundant Getter Functions

Description

These functions can be safely omitted by removing the underscore prefix from the variables they are meant to return and declaring the said variables as `public`.

Remediation

The redundant getter functions were removed from the codebase by setting the visibility specifiers of the underlying variables to `public` and properly renaming them.

VSS-03: Inexistent Sanity Check

Description

The result of `start.add(cliffDuration)` should logically be less than the result of `start.add(duration)` yet this remains unchecked in the codebase (the unit multiplier can be omitted in this case). We advise that this is imposed via a `require` check in the linked function.

Remediation

The Onther team responded by stating that the `cliffDuration` and `duration` variables are checked during initialization, however the check is a less-than-or-equal comparison (`<=`) whilst it should be a strict less-than (`<`) comparison.

Additionally, the linked logical check is incorrect as the conditional `block.timestamp >= cliff.add(duration.mul(UNIT_IN_SECONDS))` should be set to `block.timestamp >= start.add(duration.mul(UNIT_IN_SECONDS))` since the duration is meant to count from the start of the vesting period as indicated by this sanity check on the initialization.

VSS-04A , VSS-04B: Comparison Inefficiency

Description

As the linked comparisons conduct so with unsigned integers, it is possible to restrict their greater-than comparisons with zero to inequality comparisons which are also more gas-efficient.

Remediation

The comparisons were properly adjusted to their optimized inequality counterparts.

ds-math.sol (DSM)

A library that conducts wad and ray based math.

This library was not part of the audit scope as it is a widely utilized library.

Controlled.sol (CLD)

A contract implementing an owner-like controller workflow.

CLD-01: Pull Over Push Pattern

Description

Instead of directly overriding the old controller with the new one, a pattern whereby a new controller is proposed and the new controller accepts the proposal should be applied to ensure no misconfiguration occurs.

Remediation

The Onther team responded by stating that this is intended functionality as they wish to be able to revoke "ownership" rights to a contract. This is still possible after applying the Pull Over Push Pattern by simply implementing a new function called `revokeController` that nullifies the controller of the contract.

This is a paradigm that is conformed to by OpenZeppelin among other standard libraries and should be considered.

MiniMeToken.sol (MMT)

A contract implementation of a conventional ERC20 token with additional support for tracking user balances over time.

This contract was not part of the audit scope as it is a widely utilized contract.

TokenController.sol (TCL)

An interface of a token controller with the functions `proxyPayment`, `onTransfer` and `onApprove` exposed.

No findings were found at this stage of the audit.