BitBot

Security Assessment

January 15th, 2021

For :
BitBot

By :
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@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 indeed 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	BitBot
Description	A typical ERC20 implementation with enhanced features.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 2e8d513a957f38149c2965012d4aa8bf2f8013cc 2. fc4ac3f5b6a3d55c9f0765d1194e3d1eadd941f9

Audit Summary

Delivery Date	January 15th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	January 13th, 2021 - January 15th, 2021

Vulnerability Summary

Total Issues	6
Total Critical	0
Total Major	0
Total Medium	1
Total Minor	0
Total Informational	5

Executive Summary

This report represents the results of CertiK's engagement with BitBot on their implementation of the BitBot token smart contract.

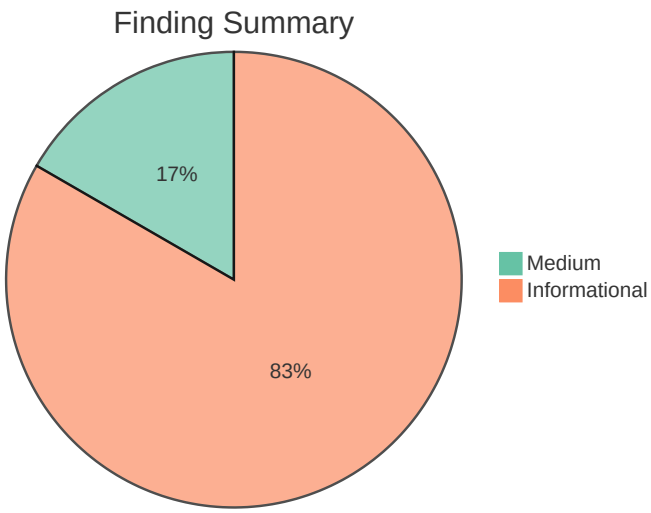
The BitBot development team opted to use the Solidity version `0.8.0`, which makes the use of the `SafeMath` library optional.

Our findings mainly refer to optimizations and Solidity coding standards, hence the issues identified pose no threat to the contract deployment's safety.

Files In Scope

ID	Contract	Location
TOK	token.sol	token.sol

Findings



ID	Title	Type	Severity	Resolved
TOK-01	User-Defined Getters	Gas Optimization	Informational	✓
TOK-02	Introduction of a `constant` Variable	Gas Optimization	Informational	✓
TOK-03	Introduction of an `immutable` Variable	Gas Optimization	Informational	✓
TOK-04	Redundant Variable Initialization	Coding Style	Informational	✓
TOK-05	Ambiguous Functionality	Logical Issue	Medium	✓
TOK-06	`external` Functions	Gas Optimization	Informational	✓

TOK-01: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	token.sol L308-L312

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The development team opted to consider our references, declared the linked variables as `public` and removed the user-defined getter functions.

TOK-02: Introduction of a `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	token.sol L312, L314

Description:

The linked statements contain either contract-level variable declarations and assignments or assignment to a literal. In both cases, the variables are never updated elsewhere in the codebase.

Recommendation:

We advise that the mutability specifier `constant` is imposed on those variables to greatly reduce the gas cost incurred by utilizing them.

Alleviation:

The development team opted to consider our references and changed the `tfees` variable to `constant`.

TOK-03: Introduction of an `immutable` Variable

Type	Severity	Location
Gas Optimization	Informational	token.sol L310, L311

Description:

The linked variables are assigned a value in the constructor function and are never changed elsewhere in the codebase.

Recommendation:

We advise that the mutability specifier `immutable` is imposed on those variables to greatly reduce the gas cost incurred by utilizing them.

Alleviation:

The development team opted to consider our references and changed the `decimals` variable to `immutable`.

TOK-04: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	token.sol L500, L501

Description:

The linked variables are redundantly initialized upon declaration, as both will contain the proper value with the statement in L503. Also, all variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The development team opted to consider our references, declared and initialized the linked variables as proposed.

TOK-05: Ambiguous Functionality

Type	Severity	Location
Logical Issue	Medium	token.sol L503-L512

Description:

The linked code segment uses the `calculateFees()` to calculate the amount of token that will be sent to the recipient and the amount that will be sent to the owner as a fee. Althought the said function is implemented and utilized as expected, the variable `amount` is used over the `amounToSend` in L511-L512.

Recommendation:

We advise to use the `amounToSend` over the `amount` variable, as two will either have equal value, in case of a feeless sender/recipient, or the former will be lesser than the latter by the fee amount.

Alleviation:

The development team opted to consider our references and changed the linked code segment as proposed.

TOK-06: `external` Functions

Type	Severity	Location
Gas Optimization	Informational	token.sol L277-L286, L295, L338, L346, L363, L370, L377, L381, L387, L401, L409, L420, L438, L456, L475, L627

Description:

The linked functions are never called by the contracts, hence can be further optimized to save gas.

Recommendation:

We advise to use the `external` attribute on the linked functions.

Alleviation:

The development team opted to consider our references and changed the visibility of the linked functions to `external`.

Appendix

Finding Categories

Gas Optimization

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

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

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.

Control Flow

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

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings 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

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.

Inconsistency

Inconsistency findings 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

Magic Number findings 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

Compiler Error findings 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.