CERTIK

Security Assessment

Artemis Token

Jun 1st, 2021

Summary

This report has been prepared for Artemis Token smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

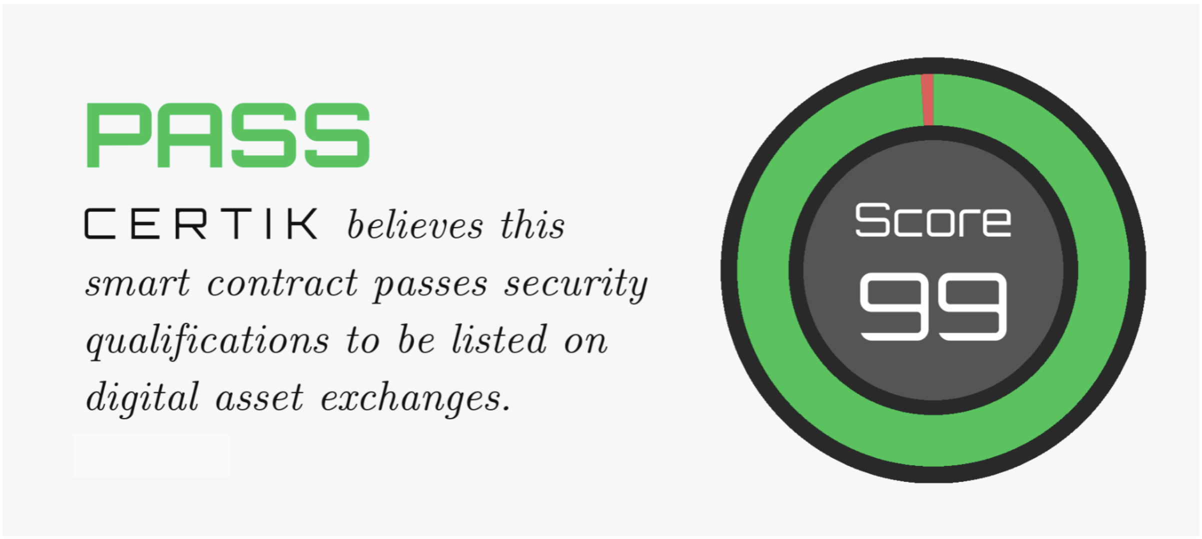
The auditing process pays special attention to the following considerations:

- 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 industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

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

No notable vulnerabilities were identified in the codebase and it makes use of the latest security principles and style guidelines. There were certain optimizations observed as well as security principles that can optionally be applied to the codebase to fortify the codebase to a greater extent.


PASS
CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*
Score
99

Overview

Project Summary

Project Name	Artemis Token
Description	A typical ERC-20 implementation with additional features.
Platform	Ethereum
Language	Solidity
Codebase	1. https://etherscan.io/address/0x28fDA76721a8077A5dE802Ab0212849B8c38429E#code 2. https://github.com/artemisguardian/artemisguardian
Commits	db691894fd826882580e0dcca00f158aeec3e824

Audit Summary

Delivery Date	Jun 01, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	ERC-20 Token

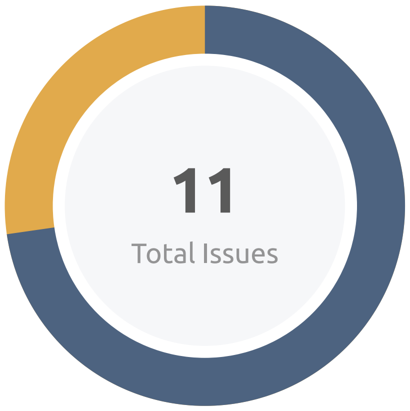
Vulnerability Summary

Total Issues	11
● **Critical**	0
● **Major**	0
● **Minor**	3
● **Informational**	8
● **Discussion**	0

Audit Scope

ID	file	SHA256 Checksum
AAV	Artemis.sol	4d1fbda5a8a4285ee90b139d0d2a3b4011ec2c9c4dc9ed68bc6a3cdcff0da9be

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**0** (0.00%)
🟨	**Minor**	**3** (27.27%)
🟦	**Informational**	**8** (72.73%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
AAV-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
AAV-02	Mutability Specifiers Missing	Gas Optimization	● Informational	✓ Resolved
AAV-03	Order of Layout	Coding Style	● Informational	ⓘ Acknowledged
AAV-04	Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call	Volatile Code	● Minor	ⓘ Acknowledged
AAV-05	Potential Over-centralization of Functionality	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
AAV-06	Redundant Contract	Gas Optimization, Coding Style	● Informational	✓ Resolved
AAV-07	Ambiguous Logic	Logical Issue	● Minor	ⓘ Acknowledged
AAV-08	Redundant `else` Clause	Gas Optimization	● Informational	✓ Resolved
AAV-09	Function Mutability Optimization	Gas Optimization	● Informational	✓ Resolved
AAV-10	Inexistent Input Sanitization	Logical Issue	● Informational	✓ Resolved
AAV-11	Unused State Variable	Gas Optimization	● Informational	✓ Resolved

AAV-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	Artemis.sol: 705	✓ Resolved

Description

The contract specifies an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

The development team opted to consider our references and locked the compiler to version `0.6.12`.

AAV-02 | Mutability Specifiers Missing

Category	Severity	Location	Status
Gas Optimization	● Informational	Artemis.sol: 1158, 1165	✓ Resolved

Description

The linked variables are assigned to only once, during their contract-level declaration.

Recommendation

We advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable.

Alleviation

The development team opted to consider our references and changed the mutability of the `MINTABLE_SUPPLY` state variable to `constant`, as `MAX_SUPPLY` was removed from the codebase.

AAV-03 | Order of Layout

Category	Severity	Location	Status
Coding Style	● Informational	Artemis.sol: 1239~1255	ⓘ Acknowledged

Description

The order of layout in the `Artemis` contract does not follow the Solidity style guide.

Recommendation

We advise to re-arrange the layout of the linked contract.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

AAV-04 | Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call

Category	Severity	Location	Status
Volatile Code	● Minor	Artemis.sol: 1275	ⓘ Acknowledged

Description

The linked `transfer()`/`transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

AAV-05 | Potential Over-centralization of Functionality

Category	Severity	Location	Status
Centralization / Privilege	● Minor	Artemis.sol: 1274~1276	ⓘ Acknowledged

Description

The linked function is meant to be used in an edge-case situation whereby the contract owner can claim the contract's remaining tokens.

Recommendation

We advise this functionality to be guarded by either a time delay to ensure that the normal course of operation of the contract has progressed.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

AAV-06 | Redundant Contract

Category	Severity	Location	Status
Gas Optimization, Coding Style	● Informational	Artemis.sol: 1013~1149	✓ Resolved

Description

The `TestDateTime` contract does not affect the functionality of the codebase and appears to be leftover from test code .

Recommendation

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

Alleviation

The development team opted to consider our references and removed the redundant code.

AAV-07 | Ambiguous Logic

Category	Severity	Location	Status
Logical Issue	● Minor	Artemis.sol: 1182~1184	ⓘ Acknowledged

Description

The linked `if` clause allows the `_teamWallet` address for an initial token withdrawal, despite the thirty-day ban.

Recommendation

We advise to revise the linked functionality.

Alleviation

The development team has acknowledged this exhibit, commenting that the linked code segment implements intended functionality.

AAV-08 | Redundant `else` Clause

Category	Severity	Location	Status
Gas Optimization	● Informational	Artemis.sol: 1189~1191	✓ Resolved

Description

The linked `else` clause is redundant, as it branches out into the default scenario.

Recommendation

We advise to remove the linked `else` clause.

Alleviation

The development team opted to consider our references and removed the redundant code.

AAV-09 | Function Mutability Optimization

Category	Severity	Location	Status
Gas Optimization	● Informational	Artemis.sol: 1181	✓ Resolved

Description

The `checkTeamWalletWithdrawalEligiblity()` function does not modify state of the contract.

Recommendation

We advise to restrict the linked function's mutability to `view`.

Alleviation

The development team opted to consider our references and added the `view` attribute to the linked function.

AAV-10 | Inexistent Input Sanitization

Category	Severity	Location	Status
Logical Issue	● Informational	Artemis.sol: 1239~1255	✓ Resolved

Description

The constructor function fails to check the values of the arguments.

Recommendation

We advise to add `require` statements, checking the input values against the zero address.

Alleviation

The development team opted to consider our references and added `require` statements, ensuring inequality of the input addresses with the zero address.

AAV-11 | Unused State Variable

Category	Severity	Location	Status
Gas Optimization	● Informational	Artemis.sol: 1158	✓ Resolved

Description

The `MAX_SUPPLY` state variable remains unused throughout the codebase.

Recommendation

We advise to remove redundant code.

Alleviation

The development team opted to consider our references and removed the `MAX_SUPPLY` state variable from the codebase.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings 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 but rather 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.

Disclaimer

About

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