ASTA Token

Security Assessment

October 14th, 2020

For :
ASTA

By :
Georgios Delkos @ CertiK
georgios.delkos@certik.org

By :
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Overview

Project Summary

Project Name	**ASTA Token**
Description	ASTA token is an ERC20 Implementation with locking functionality where owner can lock particular address tokens for time period.
Platform	Ethereum: Solidity
Codebase	Etherscan

Audit Summary

Delivery Date	**Oct. 14, 2020**
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 13, 2020 - Oct. 14, 2020

Vulnerability Summary

Total Issues	**8**
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	7

Executive Summary

The report represents the results of our engagement with the ASTA on their implementation of their ASTA token smart contract.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract deployement.

Findings

ID	Title	Type	Severity
GENT-01	Unlocked Compiler & Different Solidity Versions	Language Specific	Informational
GENT-02	Missing Natspec Comments	Coding Style	Informational
GENT-03	`external` Over `public` Function	Optimization	Informational
GENT-04	Missing `require` Statement	Volatile Code	Minor
GENT-05	Redundant Conditional	Optimization	Informational
GENT-06	Redundant `require` Statements	Optimization	Informational
GENT-07	`public` Variables Over `private`	Optimization	Informational
GENT-08	Redundant Named Variables	Optimization	Informational

GENT-01: Unlocked Compiler & Different Solidity Versions

Type	Severity	Location
Language Specific	Informational	All `pragma` statements

Description:

The compiler is not fixed and also not locked to a fixed 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 full project can be compiled at.

Alleviation

The team will be fixing the issues in the own timeframe.

GENT-02: Missing Natspec Comments

Type	Severity	Location
Coding Style	Informational	GenToken.sol

Description:

Contract code is missing natspec comments, which helps understand the code and all the functions' parameters.

Recommendation:

We advise to follow the Solidity style guide. https://solidity.readthedocs.io/en/v0.6.8/style-guide.html?highlight=natspec#natspec

Alleviation

The team will be fixing the issues in the own timeframe.

GENT-03: `external` Over `public` Function

Type	Severity	Location
Optimization	Informational	GenToken.sol L25 , GenToken.sol L38, GenToken.sol L58, GenToken.sol L63

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for functions never called from the contract.

Alleviation

The team will be fixing the issues in the own timeframe.

GENT-04: Missing `require` Statement

Type	Severity	Location
Volatile Code	Minor	GenToken.sol L38

Description:

Function doesn't check if timestamp is greater then current time, making it possible to add a lock for a previous time. Also amount should be greater than zero, as there is no point in locking zero tokens.

Recommendation:

```
require(realeaseTime > block.timestamp, "Error Message");
require(amount != 0, "Error Message");
```

Alleviation

The team will be fixing the issues in the own timeframe.

GENT-05: Redundant Conditional

Type	Severity	Location
Optimization	Informational	GenToken.sol L48

Description:

The `_lockTimes[account] != 0` is redundant, as later in the same statement code check for `_lockTimes[account] > block.timestamp`.

Recommendation:

We advise to remove the unnecessary checks.

Alleviation

The team will be fixing the issues in the own timeframe.

 GENT-06: Redundant `require` Statements

Type	Severity	Location
Optimization	Informational	ERC20.sol L54, ERC20.sol L87, ERC20.sol L88, ERC20.sol L188, ERC20.sol L169

Description:

The linked `require` statement are redundant, as the following SafeMath operations will make the exact same ones.

Recommendation:

We advise to remove the unnecessary checks.

Alleviation

The team will be fixing the issues in the own timeframe.

 GENT-07: `public` Variables Over `private`

Type	Severity	Location
Optimization	Informational	Ownable.sol L9, ERC20Detailed.sol L12-L14, ERC20.sol L20

Description:

In general, we prefer `public` variables with the automatically generated getter function over `private` ones and custom getters.

Recommendation:

Convert private variables to public and remove the custom getter functions they have implemented.

Alleviation

The team will be fixing the issues in the own timeframe.

 GENT-08: Redundant Named Variables

Type	Severity	Location
Optimization	Informational	GenToken.sol L44

Description:

The return named variables `lockTime` and `lockAmount` declared in the function signature are redundant, as the function `getLock()` directly returns the instances of the two mappings, `_lockTimes` and `_lockAmounts` respectively.

Recommendation:

Convert private variables to public and remove the custom getter functions they have implemented.

Alleviation

The team will be fixing the issues in the own timeframe.