Binance Peggy Token

Security Assessment

October 23rd, 2020

For :

Binance Peggy Token

By :

Alex Papageorgiou @ CertiK

alex.papageorgiou@certik.org

Georgios Delkos @ CertiK

georgios.delkos@certik.io

Overview

Project Summary

Project Name	**Binance Peggy Token**
Description	An ERC20 token factory that creates pegged cryptocurrency tokens on the Binance Chain.
Platform	Ethereum; Solidity, Yul
Codebase	N/A
Commits	N/A

Audit Summary

Delivery Date	**October 23rd, 2020**
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	October 16th, 2020 - October 23rd, 2020

Vulnerability Summary

Total Issues	**8**
Total Critical	0
Total Major	0
Total Medium	1
Total Minor	3
Total Informational	4

Executive Summary

The report represents the results of our engagement with the Binance on their implementation of their pegged cryptocurrency token smart contracts.

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

Files In Scope

ID	Contract	Location
AUP	AdminUpgradeabilityProxy.sol	AdminUpgradeabilityProxy.sol
BPT	BinancePeggyToken.sol	BinancePeggyToken.sol
BPF	BinancePeggyTokenFactory.sol	BinancePeggyTokenFactory.sol

Findings

ID	Title	Type	Severity	Resolved
AUP-01	Outdated Library	Logical Issue	Minor	✓
AUP-02	Outdated Implementation	Inconsistency	Informational	✓
AUP-03	EIP1967 Incompatibility	Logical Issue	Informational	✓
BPT-01	Outdated Library	Inconsistency	Minor	✓
BPT-02	Inexistent / Non-Descriptive Error Messages	Coding Style	Informational	✓
BPT-03	Incomplete `PausableToken` Implementation	Logical Issue	Minor	✓
BPT-04	Inconsistent ERC20 Support	Logical Issue	Medium	✓
BPF-01	Unused Variable Type	Gas Optimization	Informational	✓

AUP-01: Outdated Library

Type	Severity	Location
Logical Issue	Minor	AdminUpgradeabilityProxy.sol L143-L147

Description:

The `constructor` of the contract sets the implementation internally, however most proxy patterns utilize an `initialize` function call after deployment. As the `constructor` does not support also invoking an arbitrary function of the target contract via input, it is possible for a race condition to `initialize` the target implementation after the proxy is constructed.

Recommendation:

We advise that the code block is updated according to the latest version of the OpenZeppelin library, as of now v3.2.0, as the `UpgradeableProxy.sol` contract of OZ utilizes an arbitrary function call on deployment.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation, as the `BinancePeggyTokenFactory` should deploy the proxy factory to avoid race condition.

AUP-02: Outdated Implementation

Type	Severity	Location
Inconsistency	Informational	AdminUpgradeabilityProxy.sol L189-L314

Description:

The linked implementation of `AdminUpgradeabilityProxy` is outdated in comparison to the latest OZ version.

Recommendation:

We adivse that the contract code is updated according to the latest release of OZ, as of now being v3.2.0.

Alleviation:

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

AUP-03: EIP1967 Incompatibility

Type	Severity	Location
Logical Issue	Informational	AdminUpgradeabilityProxy.sol L137, L144, L210, L231

Description:

The linked code segments relate to a previous release of the OZ implementation that did not conform to the EIP1967.

Recommendation:

As it is always advised to use well-tested standards, we advise that the linked hashes are updated to reflect the requirements of the EIP1967 proxy slot standardization.

Alleviation:

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

BPT-01: Outdated Library

Type	Severity	Location
Inconsistency	Minor	BinancePeggyToken.sol L29-L77

Description:

The linked code block represents an older version of the `SafeMath` library that does not support arbitrary error messages and does not supply any error messages at all.

Recommendation:

We advise that the `library` is updated to the latest release by OZ.

Alleviation:

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

BPT-02: Inexistent / Non-Descriptive Error Messages

Type	Severity	Location
Coding Style	Informational	BinancePeggyToken.sol L105, L106, L174-L176, L323, L351, L372, L377, L433, L461, L469, L569, L607, L618, L628

Description:

The linked `require` blocks do not contain an error message specified or contain an error message that does not provide sufficient information about the condition that failed.

Recommendation:

We advise that the linked `require` statements are updated to properly aid in debugging by emitting a descriptive error message.

Alleviation:

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

BPT-03: Incomplete `PausableToken` Implementation

Type	Severity	Location
Logical Issue	Minor	BinancePeggyToken.sol L492-L553

Description:

The linked `contract` implementation attempts to implement a pausable token, however `burn` and `mint` operations can still occur thus manipulating the market.

Recommendation:

We advise that at the very least `burn` operations are guarded by the `whenNotPaused` modifier.

Alleviation:

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

BPT-04: Inconsistent ERC20 Support

Type	Severity	Location
Logical Issue	Medium	BinancePeggyToken.sol L593-L630

Description:

The linked `SafeERC20` library is outdated and reduces the support of ERC20 tokens as certain tokens such as Tether (USDT) do not return a `bool` variable on these function calls, resulting in the code segment failing.

Recommendation:

We advise that the code block is updated to the latest version by OZ that instead optionally validates the return variable only if it exists.

Alleviation:

The development team has acknowledged this exhibit but decided to support this functionality, due to the rarity of the action `reclaimToken`.

BPF-01: Unused Variable Type

Type	Severity	Location
Gas Optimization	Informational	BinancePeggyTokenFactory.sol L343

Description:

The linked assignment assigns the result of creating a new `AdminUpgradeabilityProxy` contract to an `AdminUpgradeabilityProxy` type variable, whereas it is subsequently casted to an `IBinancePeggyToken` interface on each call.

Recommendation:

We advise that it is instead casted and directly stored as an `IBinancePeggyToken` variable to reduce the number of redundant type casts.

Alleviation:

The development team opted to directly store the variable `token` as an `IBinancePeggyToken` type.

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.