Xend.Finance: Xend Token

Security Assessment

December 6th, 2020

[Preliminary Report]

For :
Xend.Finance: Xend Token

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

Sheraz Arshad @ CertiK
sheraz.arshad@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	Xend.Finance: Xend Token
Description	The codebase comprise of ERC20 implementation of Xend token which allows burning, minting and buying of XendToken.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 4cf8a749224deac304958bbc1ad54512688db9be

Audit Summary

Delivery Date	December 6th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	November 6th, 2020 - December 6th, 2020

Vulnerability Summary

Total Issues	11
Total Critical	1
Total Major	0
Total Medium	1
Total Minor	0
Total Informational	9

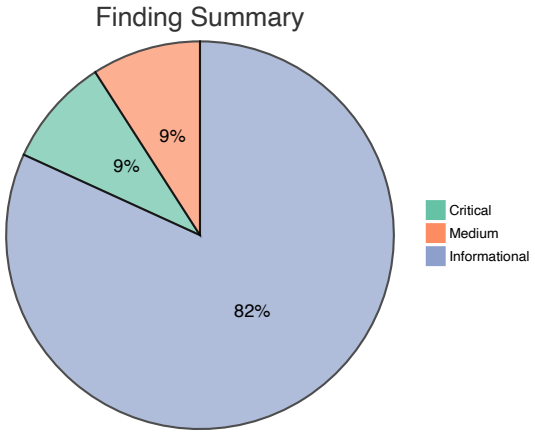
Executive Summary

This section will represent the summary of the whole audit process once it has concluded.

Files In Scope

ID	Contract	Location
ERC	ERC20.sol	ERC20.sol
IER	IERC20.sol	IERC20.sol
IXT	IXendToken.sol	IXendToken.sol
XTN	XendToken.sol	XendToken.sol
XTM	XendTokenMinters.sol	XendTokenMinters.sol

Findings



ID	Title	Type	Severity	Resolved
XTN-01	Unlocked Compiler Version	Language Specific	Informational	!
XTN-02	Imports are not used	Dead Code	Informational	!
XTN-03	`_price` can be set by any address	Logical Issue	Critical	!
ERC-01	Unlocked Compiler Version	Language Specific	Informational	!
ERC-02	Import is not used	Dead Code	Informational	!
ERC-03	Ineffecutal code	Mathematical Operations	Informational	!
ERC-04	`Transfer` event is not fired	Volatile Code	Medium	!
ERC-05	Unused function	Gas Optimization	Informational	!
XTM-01	Comparison with a literal boolean value	Gas Optimization	Informational	!
IXT-01	Unlocked Compiler Version	Language Specific	Informational	!
IER-01	Unlocked Compiler Version	Language Specific	Informational	!

XTN-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	XendToken.sol L1

Description:

The contract has 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;
```

XTN-02: Imports are not used

Type	Severity	Location
Dead Code	Informational	XendToken.sol L4, L6

Description:

The contracts from the files imported on the aforementioned lines are never used in the contract.

Recommendation:

We advise to remove the imports from the aforementioned lines to increase the legibility and quality of the codebase.

XTN-03: `_price` can be set by any address

Type	Severity	Location
Logical Issue	Critical	XendToken.sol L74

Description:

The function `SetPrice` on the aforementioned line can be called by any address setting the price of XendToken.

Recommendation:

We advise to restrict the execution of function by only the owner of the contract so that a random address could not change the price of token.

ERC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ERC20.sol L3

Description:

The contract has 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;
```

ERC-02: Import is not used

Type	Severity	Location
Dead Code	Informational	ERC20.sol L9

Description:

The contracts in the file imported on the aforementioned line are never used in the contract.

Recommendation:

We advise to remove the import on the aforementioned line to increase the legibility and quality of the codebase.

ERC-03: Ineffecutal code

Type	Severity	Location
Mathematical Operations	Informational	ERC20.sol L70-L71

Description:

The assignments on the aforementioned lines take into account the previous values of the variable. As the code resides inside the constructor and previous values are always zero so the consideration of previous can be ignored.

Recommendation:

We advise to directly the assign the values instead of adding new values to previous values as previous values are always zero.

```
_totalSupply = totalSupply;
_balances[address(this)] = totalSupply;
```

ERC-04: `Transfer` event is not fired

Type	Severity	Location
Volatile Code	Medium	ERC20.sol L72

Description:

The constructor assigns `totalSupply` as balance to `address(this)` yet does not fire corresponding `Transfer` event for the transfer. It violates the standard implementation of ERC20 tokens and can be problematic as many dApps and Blockchain explorer rely on the `Transfer` event for their operations.

Recommendation:

We advise to fire `Transfer` event inside the constructor of the contract.

```
emit Transfer(address(0), address(this), totalSupply);
```

ERC-05: Unused function

Type	Severity	Location
Gas Optimization	Informational	ERC20.sol L359

Description:

The function on the aforementioned is `internal` and never called from within the contract or any contract inheriting this contract.

Recommendation:

We advise to remove the function on the aforementioned line as it is never used.

XTM-01: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	XendTokenMinters.sol L10, L19, L26

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

IXT-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IXendToken.sol L1

Description:

The contract has 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;
```

IER-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IERC20.sol l3

Description:

The contract has 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;
```

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.