Security Assessment

Chronicle XNL

May 21st, 2021

Table of Contents

Summary

This report has been prepared for Chronicle XNL 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 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.

Overview

Project Summary

Project Name	Chronicle XNL
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/Chronicle-XNL/token/tree/5204814603b36a37b2b44e770904e1f69c93d6ea
Commits	ebb7858b2af0407a2dabafbfc001d0353d6a53fc

Audit Summary

Delivery Date	May 21, 2021
Audit Methodology	Manual Review
Key Components	

Vulnerability Summary

Total Issues	18
Critical	0
Major	0
Medium	1
Minor	8
Informational	9
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
ERC	contracts/ERC20Vestable.sol	fa5fee3b90004254cfa1e5941da6df6492b30951e33179cf3a574715d8b66e6e
IER	contracts/IERC20Vestable.sol	95218fd7e3a7e942bebfdb5c3d76455f72459c8f85bf2d78c5ccdd7c23b94315
VAC	contracts/VerifiedAccount.sol	5a00792d3ff741ff26d8df958c860ad6c487127faa5b1e435ec8f8565be92621
XNL	contracts/XNLToken.flat.sol	52c02b377324f87998b67aeba2b56e1d96de5f6109ebf3c1fa91ddfc15113014
XNT	contracts/XNLToken.sol	3cbb1f21919c1ffbf4b55724aeb1abb6e1fe15d3f9614a9138d5c24a9ad75539

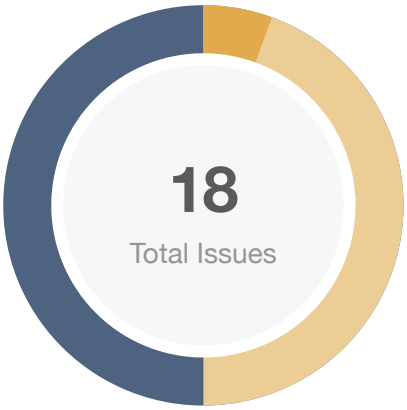
Executive Summary

Code of the project is well documented using NatSpec comments. The Vesting mechanism in particular is very well described and we haven't found any issues with it. The team was eager to address all the findings and cooperated with Certik to solve all the issues.

Only major concern we have for a project is a centralization aspect. We would like to see this issue resolved by introducing governance system or utilizing multi-sig wallet where access to it is distributed across many different trusted members. The development team agreed with us on this points and are planning to use multi-sig. For more details we suggest to check XNT-02 and XNT-03 findings.

We haven't found any major issues with the code. All the findings are below with extended description.

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Medium	1 (5.56%)
Minor	8 (44.44%)
Informational	9 (50.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
ERC-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
ERC-02	Variable tight-packing	Language Specific, Gas Optimization	Informational	Resolved
ERC-03	Should use block.timestamp	Logical Issue	Minor	Resolved
ERC-04	Unnecessary relative node_modules import	Language Specific	Informational	Resolved
ERC-05	Return value ignored	Volatile Code	Minor	Resolved
ERC-06	Explicit function return	Language Specific	Minor	Resolved
ERC-07	Multiplication on the result of a division	Mathematical Operations, Language Specific	Medium	Resolved
IER-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
VAC-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
VAC-02	Unnecessary relative node_modules import	Language Specific	Informational	Resolved
VAC-03	Redundant Statements	Language Specific	Informational	Resolved
VAC-04	Return value ignored	Volatile Code	Minor	Resolved
VAC-05	SafeERC20 not used	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
VAC-06	Functions reverts instead of returning false.	Volatile Code	Minor	Resolved
XNT-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
XNT-02	Centralization concern	**Centralization / Privilege**	**Minor**	**Acknowledged**
XNT-03	Centralization concern over functions	**Centralization / Privilege**	**Minor**	**Acknowledged**
XNT-04	Unnecessary relative node_modules import	Language Specific	Informational	Resolved

ERC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/ERC20Vestable.sol: 3	Resolved

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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation

Issue has been resolved

ERC-02 | Variable tight-packing

Category	Severity	Location	Status
Language Specific, Gas Optimization	Informational	contracts/ERC20Vestable.sol: 47~53	Resolved

Description

Variables in linked structs can be tight-packed.

Recommendation

`bool` variable can be tightpacked with any `address` variable as `address` is 160bytes and `bool` is 8bytes so two of them can be put into the same EVM slot. `uint32` variable can also be put into the same slot as `bool` and `address`.

Alleviation

Issue has been resolved.

ERC-03 | Should use block.timestamp

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ERC20Vestable.sol: 148	Resolved

Description

this means someone could schedule a vesting start in the past. We would like to know the reason of using JAN_1_2020 instead of a current block.timestamp.

Alleviation

Issue resolved.

ERC-04 | Unnecessary relative node_modules import

Category	Severity	Location	Status
Language Specific	Informational	contracts/ERC20Vestable.sol: 5	Resolved

Description

Import of `"../node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol"` is unnecessary as it can simply state `"@openzeppelin/contracts/token/ERC20/ERC20.sol"`

Recommendation

We would advise to change import to `"@openzeppelin/contracts/token/ERC20/ERC20.sol"`

Alleviation

Issue has been resolved.

ERC-05 | Return value ignored

Category	Severity	Location	Status
Volatile Code	Minor	contracts/ERC20Vestable.sol: 201~204	Resolved

Description

The linked functions invocations do not check the return value of the function call which should yield `true` in case of a proper call.

Recommendation

We would advise to check the return value of the function and create appropriate response like usage of `require` with error message.

Alleviation

Issue has been resolved

ERC-06 | Explicit function return

Category	Severity	Location	Status
Language Specific	Minor	contracts/ERC20Vestable.sol: 298~302	Resolved

Description

_getAvailableAmount() function has defined return variables as `amountAvailable` but returns explicitly `vested` variable.

Recommendation

We would recommend to either remove explicit return variable or change the variable name for return in function definition.

Alleviation

Issue has been resolved.

ERC-07 | Multiplication on the result of a division

Category	Severity	Location	Status
Mathematical Operations, Language Specific	Medium	contracts/ERC20Vestable.sol: 275	Resolved

Description

Linked function performs a multiplication on the result of a division, which can truncate:

Recommendation

We would recommend to change the order of operation from: `uint32 effectiveDaysVested = (daysVested / vesting.interval) * vesting.interval;` to: `uint32 effectiveDaysVested = daysVested * vesting.interval / vesting.interval;`

Alleviation

Issue no longer valid as per clients comments, this is intended functionality. Client comment: "The order of the operation is on purpose to allow a set individual vesting period in days. I added some examples in code comments so it should be clear why the order is chosen as it is."

IER-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/IERC20Vestable.sol: 3	Resolved

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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation

Issue resolved

VAC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/VerifiedAccount.sol: 3	Resolved

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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation

Issue has been resolved.

VAC-02 | Unnecessary relative node_modules import

Category	Severity	Location	Status
Language Specific	Informational	contracts/VerifiedAccount.sol: 6~7	Resolved

Description

Import of `"../node_modules/@openzeppelin/contracts/path/to/contract.sol"` is unnecessary as it can simply state `"@openzeppelin/contracts/path/to/contract.sol"`.

Recommendation

We would advise to change imports to `"@openzeppelin/contracts/path/to/contract.sol"`

Alleviation

Issue has been resolved.

VAC-03 | Redundant Statements

Category	Severity	Location	Status
Language Specific	Informational	contracts/VerifiedAccount.sol: 25~29, 52, 58, 63	Resolved

Description

Linked function returns unnecessary `bool ok` parameter, which is always true

Recommendation

We recommend to remove the unnecessary `bool ok` parameter.

Alleviation

Issue has been resolved.

VAC-04 | Return value ignored

Category	Severity	Location	Status
Volatile Code	Minor	contracts/VerifiedAccount.sol: 54, 59, 64	Resolved

Description

The linked functions invocations do not check the return value of the function call which should yield `true` in case of a proper call.

Recommendation

We would advise to check the return value of the function and create appropriate response like usage of `require` with error message.

Alleviation

Issue has been resolved

VAC-05 | SafeERC20 not used

Category	Severity	Location	Status
Volatile Code	Minor	contracts/VerifiedAccount.sol: 54, 59, 64	Resolved

Description

Linked functions rely on standard ERC20 functions instead of utilizing more safe SafeERC20 implementation of standard ERC20 functions.

Recommendation

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer() / approve()` functions is safely invoked in all circumstances.

Alleviation

Issue no longer valid. Linked ERC20 transfer/transferFrom/approve are only utilized for implemented token and these functions don't invoke external ERC20 calls, where SafeERC is really usefull.

VAC-06 | Functions reverts instead of returning false.

Category	Severity	Location	Status
Volatile Code	Minor	contracts/VerifiedAccount.sol: 53, 63	Resolved

Description

Linked functions calls ERC20 transfer()/transferFrom() without checking the amount, which can revert if zero.

Recommendation

We would advise to add if check for passed amount and return false when amount is zero.

Alleviation

Issue has been resolved

XNT-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/XNLToken.sol: 3	Resolved

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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation

Issue has been resolved

XNT-02 | Centralization concern

Category	Severity	Location	Status
Centralization / Privilege	Minor	contracts/XNLToken.sol: 20	Acknowledged

Description

Owner of a token contract gets all of the initial supply. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that.

Recommendation

Initial supply should be distributed to the user at start or be distributed to a handful of trusted addresses so no 1 address can hold all of the tokens at start. We would advise using multi-sig wallet that.

Alleviation

The development team has acknowledged this exhibit. Client's comment: "We plan to distribute the tokens into individual wallets by category (i.e. Partnerships, Team, Advisory) after the token is created. "

XNT-03 | Centralization concern over functions

Category	Severity	Location	Status
Centralization / Privilege	Minor	contracts/XNLToken.sol: 20	Acknowledged

Description

Owner has too much power over pausable functionality of a token. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could disturb operation of the Token

Recommendation

Mentioned functions should be called by governance or be handled by multi-sig wallet.

Alleviation

The development team has acknowledged this exhibit. Client's comment: "Obviously there needs to be an owner for the smart contract to set it up. Our position is that we are happy to have ownership for a certain time period to ensure there are no critical bugs found after the contract release and to ensure vesting periods are set correctly. After the initial period we would have the option to renounce ownership"

XNT-04 | Unnecessary relative node_modules import

Category	Severity	Location	Status
Language Specific	Informational	contracts/XNLToken.sol: 5	Resolved

Description

Import of `"../node_modules/@openzeppelin/contracts/path/to/contract.sol"` is unnecessary as it can simply state `"@openzeppelin/contracts/path/to/contract.sol"`.

Recommendation

We would advise to change imports to `"@openzeppelin/contracts/path/to/contract.sol"`

Alleviation

Issue has been resolved

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

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.

Volatile Code

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

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

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.