CERTIK

Nantrade token

Security Assessment

August 9th, 2021

Final Report

For :
Park Hoon @ Nantrade

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.

Summaries

Project Summary

Project Name	NanTrade
Description	Arbitrage Trading solution group
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	299b73f13514fbeb12a9fe453f584d8db5ea67f5

Audit Summary

Delivery Date	Aug. 9, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 7, 2021 - Oct. 8 2021

Vulnerability Summary

Total Issues	17
Total Critical	0
Total Major	2
Total Minor	2
Total Informational	14

Executive Summary

The report represents the results of our engagement with Nantrade on their Governance functionality. The initial review was conducted for three days: Aug. 10, 2021 - Aug. 13 2021 by Adrian Hetman and Alex Papageorgiou.

Several smaller issues were found during the initial audit and two major ones, a front-runner attack and vote manipulation. Both major issues and a couple of smaller ones were addressed by NanTrade and fixed with the next code revision.

Findings

ID	Title	Type	Severity
NAN-01	Inefficient greater-than comparison w/ zero	Performance	Informational
NAN-02	Incorrect version of solidity	Implementation	Minor
NAN-03	Mark external calls safe / no safe	Control Flow	Informational
NAN-04	Front-running Attack Vector	Implementation	Major
NAN-05	Comparison to a boolean constant	Performance	Informational
NAN-06	Variable tight packing	Implementation	Informational
NAN-07	Duplication of the code	Implementation	Informational
NAN-08	Vote manipulation	Logical	Major
NAN-09	Custom implementation of access control logic	Implementation	Minor

NAN-01: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	NanTradeGovernance.sol L413, NanTradeGovernance.sol L432, NanTradeGovernance.sol L192,NanTradeGovernance.sol L192, NantradeGovernance.sol L458, NanTradeGovernance.sol L494

Description:

Within Solidity, unsigined integers are restricted to the non-negative range. As such, greater-than comparisons with the literal `0` are inefficient gas-wise.

Recommendation:

Consider converting the linked comparisons to inequality ones in order to optimize their gas cost.

Alleviation:

NanTrade decided to create modifier with the same problem as described before. The team will be fixing the issues in the own timeframe.

NAN-02: Incorrect version of solidity

Type	Severity	Location
Implementation	Minor	NanTradeGovernance.sol L37, IExecutor L2, Owned.sol L2,IOwned.sol L2

Description:

The linked contracts necessitate a version too recent to be trusted. Consider deploying with 0.6.11. We do not recommend using any latest version for deployment, specially if changes were made in the optimizer or the language semantic. Version 0.6.12 made changes to optimiser that's why we do not recommend using this version.

Recommendation:

Deploy with any of the following Solidity versions:

- 0.5.11 - 0.5.13,
- 0.5.15 - 0.5.17,
- 0.6.8,
- 0.6.10 - 0.6.11. Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Alleviation:

The team decided to stay with current version of solidity i.e. `0.6.12`

NAN-03: Mark external calls safe / no safe

Type	Severity	Location
Control flow	Informational	NantradeGovernance L379

Description:

`IExecutor(proposals[_id].executor).execute(_id, forRatio, againstRatio, quorumRatio)` function call is an external function call. While re-entrancy is not possible for the `execute` function of the `NanTradeGovernance` contract, other unintended re-entrancy interactions may occur by the external contract should the quorum of NanTrade not have vetted its code properly.

Recommendation:

We advise that a comment is inserted in the preceding external call line that explains the call is safe as it has been voted on and validated by the quorum of NanTrade.

Alleviation:

Issue was resolved.

NAN-04: Front-running Attack Vector

Type	Severity	Location
Implementation	Major	NanTradeGovernance L377

Description:

`tallyVotes` function (L390 - L405) which is executed in the `execute` function (L370 - L383) is marked as a `public` function. During the `execute` function, `tallyVotes` changes the `proposals[_id].open` property from `true` to `false`.

This param is used in the `proposalEnded` modifier, leading to an `ERR_NOT_OPEN` throw if the proposal is closed and reverting the transaction. `tallyVotes` can be called by anyone who knows the `id` of the proposal to close it and thus stoping further execution of the proposal.

This attack vector is especially exploitable via a front-running attack whereby one inspects the transaction mempool of Ethereum, detects an `execute` contract call and invokes `tallyVotes` beforehand with a higher gas fee.

Recommendation:

`tallyVotes` can be marked as a `private` or `internal` function thus eliminating the potential for a DoS-type attack on a proposal's execution.

Alleviation:

Issue was resolved but some optimization can be still done on `execute` function. `tallyVotes()` can be made internal and already-calculated `forRatio` and `againstRatio` can be passed to the function directly without the need of calculating them again.

NAN-05: Comparison to a boolean constant

Type	Severity	Location
Performance	Informational	NanTradeGovernance L201

Description:

The `onlyVoter()` modifier uses a boolean value to compare with a boolean literal.

Recommendation:

Boolean values can be used directly and do not need to be compared to `true` or `false`.

```
modifier onlyVoter() {
      require(voters[msg.sender], "ERR_NOT_VOTER");
      _;
  }
```

Alleviation:

Issue was resolved by removing this modifier and removing `revokeVotes()`.

NAN-06: Variable tight packing

Type	Severity	Location
Implementation	Informational	NanTradeGovernance L54-L68

Description:

Variables in the struct `Proposal` can be tightpacked.

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. `uint256 start` and `uint256 end` could be changed to `uint128` and tightpacked together as block number won't ever be larger than maximum of `uint128`.

Alleviation:

Problem partially resolved. `bool` and `address` are tight packed but `uint256 start` and `uint256 end` are still not changed to `uint128`. The team decided not to change `uint256` start and end variables to keep them the same type as timestamp.

NAN-07: Duplication of the code

Type	Severity	Location
Implementation	Informational	[NanTradeGovernance L210-L233]()

Description:

modifiers `proposalNotEnded` and `proposalEnded` share the same code in the first `require` which could be put into its own modifier code. This can cause some confusion and potential issues when one code block is updated and other one not.

Recommendation:

```
require(
            proposals[_id].start > 0 && proposals[_id].start < block.number,
            "ERR_NO_PROPOSAL"
        );
```

This code block can be extrapolated and putted into separate modifier called `validProposal`.

Alleviation:

Issue was resolved.

NAN-08: Vote Manipulation

Type	Severity	Location
Logical	Major	[NanTradeGovernance L426-L444, L517-L527]()

Description:

The `unstake` and `revokeVotes` functions affect the `totalVotes` variable of the contract, however they do not adjust already-voted-on proposals. This leads to proposals reporting invalid quorums and total votes available as they are re-set on each vote. This is especially exploitable in case a proposal's expiration is before the vote lock mechanism, meaning a double-vote can occur without losing balance.

To replicate this issue, simply `stake` some new tokens for 2 different accounts. Have account A vote for a proposal and then instantly revoke his votes and have account B vote against a proposal. The `totalVotesAvailable` and `quorum` variables of the proposal will be incorrect, leading to invalid calculations on all functions relating to a proposal's acceptance.

Recommendation:

We advise two things. First, an account's votes should be locked until the expiration date of the proposal and ensured to be the maximum expiration of all ongoing proposals voted on.

Secondly, a `require` check should also be imposed on `revokeVotes` that prevents revocation in case a proposal is in progress.

Alleviation:

Issue was resolved.

NAN-09: Custom implementation of access control logic

Type	Severity	Location
Implementation	Minor	NanTradeGovernance L39

Description:

`Owned.sol` contract seems to implement it's own logic for access control instead of relying on openzeppelin's `Ownable.sol` contract

Recommendation:

We advise using Openzeppelin implementation of Ownable.sol contract instead.

Alleviation:

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