THX Network Security Analysis

by Pessimistic

This report is public.

Published: July 16, 2021

Abstract

In this report, we consider the security of smart contracts of THX Network project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of THX Network smart contracts. We performed our audit according to the procedure described below.

The initial audit only showed several issues of low severity.

After the initial audit, the code base was updated. Most of the issues were fixed. For the rest of the issues, developers provided comments.

The documentation for the project is incomplete.

General recommendations

We recommend simplifying folder structure of the project, enabling compiler optimization, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with THX Network project on a public GitHub repository, commit b5587b48112e3b319212aef06264c1d38fbe1cad.

The repository contains README.md file with the description of the project. Some parts of the code have NatSpecs. However, we consider this documentation incomplete.

The project compiles with warnings. All tests pass, the code coverage is 27.37%.

The total LOC of audited sources is 1499.

Code base update

After the initial audit, the code base was updated. For the recheck we were provided with commit e352b0d6514e3391e86bb40fd2059fafb9e4f565.

The documentation was improved. Also, developers added new tests and therefore improved the code coverage. However, the coverage is not measured.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

The audit showed no issues of medium severity.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Code logic (fixed)

In **util/BasePoll.sol** file, `voteValidate()` function has unused `_agree` parameter. Since the function is declared as `virtual`, it can be overridden the way that it invalidates certain votes and therefore can be used for censorship.

We recommend removing `_agree` parameter.

The issue has been fixed and is not present in the latest version of the code.

Code quality

- The project's architecture is inspired by Diamond design pattern. **RewardPoll** and **WithdrawPoll** facets include external functions, that are supposed to be called only from other facets. However, in the current implementation they can be called directly.

  Though it does not cause an issue since **RelayDiamond** manages incoming transactions, we recommend adding `require(msg.sender == address(this))` check to prevent direct calls from inner facets.

  The issue has been fixed and is not present in the latest version of the code.

- Facets **05-Withdraw** and **06-Reward** share storage and implement identical `_createWithdrawPoll()` functions, which compromises code modularity of the Diamond Standard.

  > *Comment from developers: an issue has been created: https://github.com/thxprotocol/modules-solidity/issues/6.*

- In **06-Reward/Reward.sol** and **06-Reward/RewardPoll.sol** files, there are magic numbers used for enabling/disabling the rewards. Using magic numbers is considered a bad coding practice, especially when used more than once.

  We recommend implementing separate functions to change the rewarding state.

  The issue has been fixed and is not present in the latest version of the code.

- In `_createWithdrawPoll()` function of **Withdraw** contract, the returned value is never used. Consider removing it.

- In **AssetPoolFactory/AssetPoolFactoryFacet.sol** at line 22, the loop counter variable `i` in not initialized.

  The issue has been fixed and is not present in the latest version of the code.

- Consider declaring `public` functions as `external` where possible.

  The issue has been fixed and is not present in the latest version of the code.

Gas consumption (fixed)

- In **05-Withdraw/WithdrawPoll.sol** file at line 66, consider using custom error message parameter of `safeMath.sub()` function instead of explicit check.
- Consider declaring the address of `unlimited` token account as `immutable` in **util/TokenUnlimitedAccount.sol**.

The issues have been fixed and are not present in the latest version of the code.

Project management

- The code coverage is low and is not measured. Key functionality is not covered.

  The issue has been fixed. New tests were added, the code coverage was improved.

- The project compiles with warnings, the optimization is disabled.

> *Comment from developers: The OpenZeppelin **Ownable** and **ERC20** contracts implement visibility for contract constructors. They are ignored by our current compiler, and this causes a warning. OpenZeppelin already has ^0.8.0 contracts available and we will move to the 0.8.0 compiler when the diamond-2 package releases a new version containing the ^0.8.0 update which is already available in the repo. This should resolve the warnings.*
> *AccessControlMock contract throws warnings due to unused function parameters which is correct. The mock contract overrides an existing interface implemented by other contracts that do use the function parameter. We are required to follow this interface and accept the warning it causes, since the mock does not do anything with the param by design.*

Incomplete documentation (fixed)

The provided documentation is incomplete. The repository has a README file with a rough description of the project. Some parts of the code do not have NatSpecs.

Proper documentation should explicitly describe the purpose and behavior of the contracts, their interactions, and main design choices to simplify the development process and make auditing process more effective.

The documentation was improved.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

July 16, 2021