DeFireX

DeFireX Security Analysis

by Pessimistic

This report is public.

Published: January 5, 2021

Abstract

In this report, we consider the security of smart contracts of DeFireX project. Our task is to find and describe security issues in 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 smart contracts of DeFireX project. We performed our audit according to the procedure described below.

The initial audit showed two critical issues, several medium severity issues, and issues of low severity. The project has no documentation, tests, or deployment scripts.

After this audit, we discussed the results with the developers. The code was updated to the latest version. The functionality, that resulted in no replay protection issue, was removed. The developers also provided an explanation for uninitialized variable issue, so we marked it as not an issue.

The contracts are upgradeable and were deployed on earlier commit. However, earlier version of the code and upgrade process are beyond the scope of the audit.

General recommendations

We recommend adding the documentation, implementing tests and deployment script, and using automated security tools as essential part of the development process.

Procedure

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

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
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 inspect the specification and check whether the logic of smart contracts is consistent with it.
  - 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.

Project overview

Project description

In our analysis we consider smart contracts of DeFireX project on GitHub repository, commit 9b5f832a4576826ab8c1dcdd6ca6a9e81b76b047.

The contracts were deployed on earlier commit which is beyond the scope of the audit.

The scope of the audit included the following contracts (with their dependencies):

- **deposits/DfDepositTokenV3.sol**
- **deposits/DfTokenizedDeposit.sol**
- **deposits/DfFinanceDeposits.sol**
- **deposits/DfWalletDeposits.sol**

The project has no documentation and no tests.

Latest version of the code

For the recheck we were provided with commit 3bf1ddcc733c2440c52c497c98d58f905cec57fb.

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.

No replay protection (fixed)

In **token/ERC20/ERC20Delegatable** contract, `delgateWithSign()` and `undelgateWithSign()` functions use `onlyVerified` modifier to verify user's signature. However, they do not have any nonce or hash-based protection to make a call unique.

As a result, once a user made a call, anyone can repeat this call.

Consider including nonce to these functions to make sure that every call written to blockchain is unique.

Functions `delgateWithSign()` and `undelgateWithSign()` were removed from the contract in commit a0ba51198f7043610f8a3d0c9aba8cac99fa8343.

Uninitialized variable (not an issue)

In **deposits/DfTokenizedDeposit** contract, `token` variable at line 65 is not initialized.

This issue could be easily found with tests or automated tools, e.g. Slither.

> *Comment from developers: We use upgradable smart contracts, and they are already deployed. Also, the state variables are already initialized. In the current version, we use the `createStrategyDeposit()` function (can only be called once) — line 675 of the current **DfTokenizedDeposit** implementation — https://etherscan.io/address/0xd82c46e783cf0f4c260332e29be60aace3d59049#code.*
>
> *We removed unnecessary logic due to the 24KB contract size limit (EIP170).*

Medium severity issues

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

No tests and deployment script

The provided code does not contain tests. Testing is crucial for code security and audit does not replace tests in any way.

We highly recommend both covering the code with tests and making sure that the test coverage is sufficient.

There is also no deployment script. The contracts deployment does not seem trivial. Bugs and vulnerabilities often appear in deployment scripts and severely endanger security of the system.

Deployment procedure is especially crucial for upgradable contracts, as this part of the project is very error prone. We highly recommend testing deployment scripts carefully.

No documentation

The project has no documentation. As a result, it is sometimes unclear for the auditor what is the intention of the code, is its behavior correct, and whether the architecture of the project is appropriate.

Considering the complexity of the project, the documentation is critically important not only for the audit but also for development process. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

Low severity issues

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

Code quality

- In **deposits/DfWalletDeposits.sol** at line 15, **DfWallet** contract should implement **IDfWallet** interface.
- In **deposits/DfFinanceDeposits.sol**, `compClaimedinUSD` property of `UserData` struct at line 53 is never used.
- In **deposits/DfTokenizedDeposit** contract, there are multiple hardcoded address of other contracts are used. Consider providing the address via initializer or ENS.
- Consider declaring functions as `external` instead of `public` where possible.

Gas consumption

- In **DfWallet** contract, consider claiming COMP directly to a user with `claimComp(address holder, CToken[] memory cTokens)` call of **IComptroller** contract to reduce gas consumption.
- In **DfTokenizedDeposit** contract, the usage of `unit64` is appropriate only in `ProfitData` struct. In other cases, it increases gas consumption. Consider using `uint` instead.

Project management

Config files (**truffle.js**, **networks.js**) and **package-lock.json** are added to **.gitignore** file.

We recommend adding them to the repository.

ERC20 incompliance

IDfDepositToken and **IToken** interfaces have `approve()` function without return value.

According to EIP-20, `approve()` function returns a value of `bool` type:

```
function approve(address _spender, uint256 _value) public returns (bool
success)
```

Notes

Overpowered owner

In **deposits/DfDepositToken** contract, the owner can mint tokens to any address or burn them without any limitations.

> *Comment from developers: We cannot remove the mint because owner is **DfTokenizedDeposit** – it is our protocol logic. Remove the burn will increase gas consumption for user (must send approve before burn).*
>
> *There is no logic to transfer token ownership in this contract.*

*However, **DfTokenizedDeposit** contract is upgradeable, and therefore its logic may change. But the upgrade process is beyond the scope of this audit.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

January 5, 2021