xDao Security Analysis

by Pessimistic

Published: August 30, 2021

Abstract

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

Disclaimer

Summary

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

The initial audit showed several issues of medium severity, including Non-ERC20 token, and Change of subscription variables. Also, several low severity issues were found.

The project has insufficient documentation.

After the initial audit, the code base was updated. All the issues were either fixed or commented.

General recommendations

We recommend improving the documentation.

Project overview

Project description

For the audit, we were provided with xDao project on a public GitHub repository, commit cb3e831e930d19a9aebbc12be84363a0412c9cb2.

The documentation for the project is stored on GitBook platform. However, the code is lacking NatSpecs.

The project compiles with warnings.

All 26 tests pass, the coverage is 100%.

The total LOC of audited sources is 859.

Token details:

Name:	xDAO Token
Symbol:	XDAO
Decimals:	18
Total supply:	10e9 * 10e18

Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with commit 8feabcfcbf4f03f666448f5500f54e3daad3e42e.

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.

Non-ERC20 token

According to EIP-20, tokens transfer is considered a basic functionality. However, Dao contract reverts on `transfer()` or `transferFrom()` calls. Therefore, the token is not ERC20, and declaring it as ERC20 is confusing.

> *Comment from developers: We inherit the **ERC20** contract from `openzeppelin` to avoid possible errors during self-implementation. The DAO architecture prohibits the free transfer of GT tokens, so we reverted the `transfer()` and `transferFrom()` functions.*

Change of subscription variables

The owner of **Factory** contract can change `monthlyCost` and `freeTrial` values at any moment without notifying users. As a result, the users of the project can get results they do not expect when paying for subscription or using free trial.

Since `monthlyCost` is transferred directly to the owner, `changeMonthlyCost()` function can be exploited to front-run users who called `subscribe()` function.

Consider applying changes after a delay so that users could see the price change before calling `subscribe()` function or adding a check that the subscription cost did not increase.

> *Comment from developers: The owner of the **Factory** contract will be a DAO with the most decentralized form of decision making. We estimate the frequency of changes in the subscription price and the duration of the free trial as low, and with full notification of people via social networks and the website. Therefore, we do not introduce a delay to change these parameters.*

Insufficient documentation

The project has a documentation. However, this documentation does not cover the whole project. E.g., the documentation does not describe the purpose of `Adapter` entity, the intention of `withdraw()` function, etc. Also, the code is lacking NatSpecs.

Proper documentation should explicitly describe the purpose and behavior of the contracts, their interactions, and main design choices. It is also essential for any further integrations.

> *Comment from developers: At the moment, no `Adapter` has been written. We left this opportunity for the implementation of contracts of the following type in the future:*
> *The address burns its LPs, the `Adapter` takes out a fraction of this address from Sushi Farms using the `withdraw()` function and sends the Sushi LP to this address.*

Low severity issues

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

Code quality

- CEI pattern is violated in several places, e.g. `executePermitted()` function of **Dao** contract, `createLp()` function of **Shop** contract, `subscribe()` function of **Factory** contract. We highly recommend following CEI pattern since it does not increase the cost of execution for honest users but improves usage predictability.

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

- Consider declaring the visibility of `executedVoting` variable explicitly at line 64 of **Dao** contract.

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

- Consider adding `chainId` parameter when calculating `TxHash` in `execute()` function of **Dao** contract.

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

- In **Factory** contract, the left part of the condition at line 37 is redundant since the right part of the condition includes this case, i.e. if `subscriptions[_dao] == 0`, then `subscriptions[_dao] < block.timestamp` will always evaluate to `true`. Since `subscriptions[_dao]` value is read from storage, this redundant check also increases gas consumption.

  The same issue applies to the check at lines 293–295 of **Dao** contract.

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

- `setFactory()` function of **Shop** contract does not check if provided address is correct. Addresses for **Shop** and **Factory** contracts can be pre-calculated at deployment stage.

  > *Comment from developers: We do that only once, so we skip these checks.*

Code logic (fixed)

In `burnLp()` function of **Dao** contract, the check at line 336 does not consider the case when different adapters are used within the same pool. In such a case, the share from the same pool will be withdrawn multiple times.

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

Gas consumption (fixed)

- Consider saving `publicOffers[_dao]` value from storage to a `memory` variable in `buyPublicOffer()` function of **Shop** contract to reduce gas consumption.
- Consider declaring `address` parameter of `Received` event at line 603 of **Dao** contract as `indexed` to allow event filtering.
- In **Dao** contract, variables `executedVoting` and `executedPermitted` are not used anywhere. Consider using events instead.
- In **Dao** contract, the checks at lines 439, 449, 459, and 467 are redundant since they are performed inside `add()` and `remove()` functions of **EnumerableSet** library. Consider checking the returned values of `add()` and `remove()` functions instead.

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

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

August 30, 2021