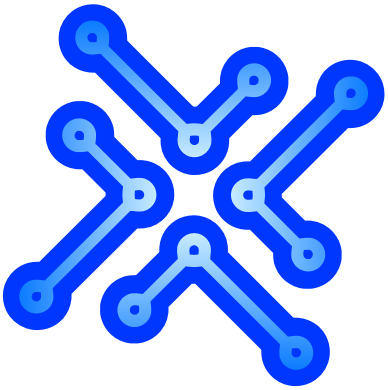

XDAO

XDAO Security Analysis

by Pessimistic

This report is public

August 30, 2023

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

The audit does not give any warranties on the security of the code. A single 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, a security audit is not investment advice.

Summary

In this report, we considered the security of XDAO smart contracts. We described the audit process in the section below.

The audit showed several critical issues: Adding allocation for another's vesting, The usage of another's tokens, and The usage of another's tokens#2. The audit also revealed two issues of medium severity: Incorrect accounting of multiple allocations and Privileged roles. Moreover, several low-severity issues were found.

After the initial audit, the codebase was updated. The developers fixed all critical issues: Adding allocation for another's vesting, The usage of another's tokens, and The usage of another's tokens#2. They also fixed several issues of low-severity and fixed or commented on all the issues of medium severity: Incorrect accounting of multiple allocations and Privileged roles. One new low-severity issue was found. The developers provided a comment on this issue. The number of tests increased.

The overall code quality is good. The codebase is roughly covered with tests.

General recommendations

We recommend fixing the remaining issues and improving NatSpec coverage.

Project overview

Project description

For the audit, we were provided with XDAO project on a public GitHub repository, commit 8ef7d69d5927dc5001989c719d539855a0130c28.

The scope of the audit included:

- **contracts/modules/CrowdfundingModule.sol**,
- **contracts/modules/DaoManager.sol**,
- **contracts/modules/DaoVestingModule.sol**,
- **modules/PrivateExitModule.sol**.

The documentation for the project included a private Notion doc.

All 17 tests passed successfully. The code coverage was 100%.

The total LOC of audited sources is 916.

Codebase update

After the initial audit, the codebase was updated, and we were provided with commit 07b12d4a0847b58da6ceba17082c083a4d8f18d4.

This update contained fixes and comments for all issues. However, we found the new issue of low-severity which was commented on by the developers.

All 21 tests passed. The code coverage was 100%.

Audit process

We started the audit on August 7, 2023, and finished on August 15, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- If the funds stored in contracts are safe;
- If the calculations are performed correctly;
- Possible gas optimizations.

We scanned the codebase with Semgrep rules for smart contracts and provided the developers with the result. However, we were not able to scan the project with the static analyzer Slither and our plugin Slitherin due to the unresolved issue in the Slither.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

We initiated recheck on August 28, 2023, and completed on August 29, 2023.

All developer-provided fixes reviewed.

We re-run tests and calculated the code coverage.

Finally, we updated 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 project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Adding allocation for another's vesting (fixed)

The `addAllocation` function allows adding a new allocation to an existing vesting. This function allows specifying to which DAOs vesting the allocation should be added. However, it is only checked that the `msg.sender` is `CrowdfundingModule` or the `msg.sender` is a DAO in XDAO ecosystem. This means that one DAO can add an allocation to another DAO's existing vesting. This allows a way to steal the tokens from the contract in the `addAllocation` function of the **DAOVestingModule** contract.

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

C02. The usage of another's tokens (fixed)

In the **CrowdfundingModule** contract, a malicious user can create crowdfunding, setting another DAO's token as a sale token. The contract does not check if the DAO has provided tokens that it is selling. Thus, the malicious user can steal all tokens that are present in the `CrowdfundingModule` contract's balance. Note that the malicious user will need to manage a DAO in the XDAO ecosystem to perform this attack. However, they can register a new DAO from the XDAO factory.

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

C03. The usage of another's tokens#2 (fixed)

A similar attack to C01 is possible in the **DaoVestingModule** contract. The malicious user can create a new vesting, setting another DAO's token as a vesting token. By doing this, the attacker can steal all tokens that are present in the contract balance in the **DaoVestingModule** contract.

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

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Incorrect accounting of multiple allocations (fixed)

The user can receive multiple allocations for the same vesting. However, if the second allocation is received after some time has passed, and the user has already released some part of the first allocation, the user will receive fewer tokens than they should. This is because of how the contract calculates the releasable token amount. The user will lose `(lastClaimed - startTime) * tokenAmount` amount of tokens.

We recommend storing how many tokens a user has already claimed and giving the user `currentReleasable - alreadyClaimed` amount of tokens, where `currentReleasable` is the token amount which corresponds to how many token the user should receive at the current timestamp (given the user had not claimed any tokens). This will also solve the rounding issue described in L08 in the **DaoVestingModule** contract.

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

M02. Privileged roles (commented)

The system relies heavily on the admin role. The admin can tweak some contract settings (e.g., update fees) and also upgrade the logic of the proxy contracts.

There are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the private key for this role becomes compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: In the future, these roles will be managed by the DAO.*

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Early LP token burn risk (commented)

If the DAO has created the offer for private exit, there is a possibility that anyone can call the burnLP function before the DAO decides to exit. This means that the burning of LP tokens can occur earlier than intended by the DAO. Consider calling burnLp inside the `closeSale` function if the crowdfunding token is `LP` and transfer tokens if otherwise in the `burnLp` function of the **CrowdfundingModule** contract.

> *Comment from the developers: We fully agree with the comment however we believe that DAO intends to burn LP as soon as privateExitOffer was created. Also to preserve flexibility and usability (for example, when all tokens have been redeemed and there is no need to create a privateExitOffer to avoid creating unnecessary voting) we would like to keep closeSale and burnLp functions separated. However, we decided to remove the _isSendTokensBack argument in closeSale to prevent locking the tokens in Crowdfunding tokens.*

L02. Extra reading from storage (fixed)

The `saleIndexes[_dao]` variable is read multiple times from storage at line 271 in the `_editSale` function of the **CrowdfundingModule** contract. Consider reading it once to the local variable to reduce gas consumption. The same case is in the `releasable` function of the **DaoVestingModule** contract (`vesting.start`, `vesting.duration` variables) at lines 249.

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

L03. Gas optimization (fixed)

The `Sale` structure fields can be reordered so that the `bool` and `address` fields will be next to each other and will be written to the same storage slot to reduce gas consumption. It will reduce gas consumption at lines 35–49 in the **CrowdfundingModule** contract.

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

L04. No event (commented)

Consider emitting events in the `createPrivateExitOffer` and the `disablePrivateExitOffer` functions in the **PrivateExitModule** contract.

> *Comment from the developers: Because of the low severity of this issue, we would like to leave it to avoid redeployment.*

L05. No modifier (commented)

The `createPrivateExitOffer` function of the **PrivateExitModule** contract does not have the `onlyDao` modifier, which allows anybody to create a private offer with a fake DAO. Moreover, the specified fake DAO can return any token address when `dao.lp();` is called. Note that at the moment, there are no possible attacks using this issue. However, this can potentially create an attack vector in the integration contracts. Thus, we recommend restricting non-DAOs from creating a private offer.

> *Comment from the developers: Because of the low severity of this issue, we would like to leave it to avoid redeployment.*

L06. Redundant approval (commented)

The approval from the **PrivateExitModule** contract to the **LP** contract for spending `LP` tokens is not required in the code at line 122 in the `privateExit` function of the **PrivateExitModule** contract.

> *Comment from the developers: Because of the low severity of this issue, we would like to leave it to avoid redeployment.*

L07. Redundant flag (commented)

The `regularFeeRate` and `discountFeeRate` variables both utilize the common `setFee` setter, located at line 125 (This setter is exclusively called by the `DEFAULT_ADMIN_ROLE`). Importantly, these variables remain static throughout the code, with no alterations elsewhere. When executing the `buy` function, users possess the option to select one of these fees through the use of the `_isRegularFee` flag. Given this arrangement, there exists no practical rationale for segregating these fees. This is due to the fact that users will invariably choose the lesser of the two fees at line 385 in the `buy` function of the **CrowdfundingModule** contract.

> *Comment from the developers: We completely understand that investor has the ability to pay a lower fee than they are supposed to. We are expecting that most investors will use the XDAO interface, where there will be no option to select the amount of fee. This feature is needed to provide discounts based on data not stored onchain.*

L08. Rounding error (fixed)

If a user frequently releases tokens from their vesting schedule, there is a risk of the value being reset or rounded down due to minimal time differences between the last update and the current time interval. Consequently, the user may lose access to the remaining balance of tokens during that specific interval, preventing them from claiming the full designated amount in subsequent intervals at lines 230–231 in the `releasable` function of the **DaoVestingModule** contract.

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

L09. Uninitialized logic contract (fixed)

Consider initializing logic contracts by invoking `_disableInitializers()` in the constructor. There is a possible attack with the initialization of the logic contracts of the `UUPS` proxies. Even though this issue is not present in the current version of the OpenZeppelin contracts, it is still recommended to keep logic contracts initialized.

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

L10. Overallocation risk (commented)

The DAO can potentially allocate more tokens to claimers in the **DaoVestingModule** contract than what is available in the `remainingTokenAmount` variable (balance in vesting). This means that the sum of all DAO allocations can exceed the available tokens. If the DAO fails to replenish the balance promptly through `fillLpBalance` or `fillTokenBalance` functions, some claimers may not receive their expected share from the vesting.

> *Comment from the developers: In case DAO allocates more tokens than are available (remainingTokenAmount), all claimers will be notified about the need for DAO to replenish the balance through the XDAO interface.*

This analysis was performed by Pessimistic:

Yhtyyar Sahatov, Security Engineer
Daria Korepanova, Senior Security Engineer
Irina Vikhareva, Project Manager

August 30, 2023