Superdao
Security Analysis

by Pessimistic

This report is public

September 13, 2022

Abstract

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

No critical security issues were discovered. The initial audit showed several issues of medium severity: Loss of unused ether, Token price error possibility, Incorrect claiming condition and Overpowered role. Also, several low-severity issues were found.
The overall code quality is above average.

After the initial audit the codebase was updated. All medium issues were fixed and the most of low severity issues were addressed. However, one new issue regarding the fix was discovered.

General recommendations

We recommend fixing the remaining issue. We also recommend improving NatSpec coverage and implementing CI to analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Superdao project on a private repository, commit `b7d97cc4bfccf479efea9fc6bc0ed3ef6a35d588`.

The documentation for the project includes documents on system architecture.

All 81 tests pass successfully. The code coverage is 86.41%.

The total LOC of audited sources is 2009.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit `35877c53f4a02ec3bc7cc6152c149468a10e40fa`. This update includes only fixes for most of the issues mentioned in the initial audit.

With this update we found one new code issue.

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 the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Inter alia, we verify that:

- No standard Solidity issues are present in the codebase.
- Meta-transactions are secure, and the signatures comply with EIP-712.
- UniswapV3 integration is exucuted properly.
- Access control is implemented correctly.
- Contract's upgradeability is secure.

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.

The audit showed no critical issues.

Medium severity issues

Medium 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. Loss of unused Ether (fixed)

On executing the `executeSingle` function of **CallForwarder** contract (both regular and meta-transaction versions), the `msg.value` may contain more Ether than specified in `req.value`. In that case (since only `req.value` is transferred to a payable function inside the `functionCallWithValue()` of the **Address** library), the remaining amount of Ether will remain on the **CallForwarder** address and anyone can withdraw it with a call to `executeSingle`. We recommend returning any unused Ether instantly.

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

M02. Token price error possibility (fixed)

At line 148 of the `_getSqrtPriceX96` function of **UniswapV3Oracle** contract there is a typecasting for `twapInterval_` from `uint32` type to `int32`. If the value exceeds half of `type(uint32).max/2`, such typecast leads to an overflow and wrong result.

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

M03. Incorrect claiming condition (fixed)

There is a bug at line 82 in `_claim` function of the **ERC721BaseSale** contract. Following the code claiming logic (line 91, claims increament), the correct check here should ensure that `claimLimit > claims[_msgSender()]`, not `claims[_msgSender()] > claimLimit`. We recommend fixing the bug and adding tests for this case.

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

M04. Overpowered role (addressed)

The `upgradeAppImpl` function of **Kernel** contract allows users with `KERNEL_ADMIN` privileges to set arbitrary contract address for an application. If a malicious user takes control over one of `KERNEL_ADMIN` accounts, he could remove rights from other admins and exploit app users. For example, an admin can call `upgradeAppImpl()` to set a malicious contract as one of the apps so that every time a user calls the app, the contract transfers some tokens or Ether to itself.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Within the protocol, the KERNEL_ADMIN role is owned by the SUDO and RELEASE_MANAGER entities which belong to project maintainers. This is done intentionally for quick iterations, as maintainers have to update and deliver code to users.*
>
> *If a user (DAO) wants autonomy, these roles might be moved to their management contract (Safe, Timelock) and thus all decisions to change implementations will be by consensus.*

M05. Issue with batch sending of Ether (new)

Since batch transactions in the **CallForwarder** contract rely on the `executeSingle` function, the calls with multiple transactions containing Ether transfer will fail. Suppose you send two transactions with Ether. The second transaction will fail because the `_refund` call (added to fix the M01 issue) in the `executeSingle` function from the first transaction will return the remaining Ether (intended for the second transaction) to the user.

We recommend extracting the actual call with a transfer into an internal function and implementing two separate refund solutions (or `msg.value` checks) for the cases of `executeBatch` and `executeSingle`.

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. Code quality: hardcoded address (addressed)

In the `setEvent` function of the **Kernel** contract, the operator address is hardcoded at line 119. All such addresses should be marked as constants.

> *The developers have planned fixing this issue in the further update.*

L02. Redundant check (fixed)

The value of the `rest` variable will always be less than the value of the `available` variable at line 124 in **ERC721BaseSale** contract.

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

L03. Missing array length check (fixed)

In the **AdminController** contract, two arguments of the `batchMint()` (`to` and `tierValues`) are arrays. Consider adding a check to ensure their lengths are the same. This issue also applies to the `initialize` function of the `ERC721WhitelistClaim` contract.

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

L04. Missing constant reference (fixed)

The length of the array at line 20 in **BaseStorage** contract should equal `ACL_SLOT_SIZE` .

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

L05. Unused variable (addressed)

The `AppInfo` structure is utilized within `_appInfo` internal mapping. The `isActive` flag of this structure is initialized in the **AppManager**, however this flag is never read in the project.

> *The developers have planned fixing this issue in the further update.*

L06. Double versioning (not fixed)

Note, that **CallForwarder** deployment uses two versions: first contract version is set via `__with_semver` and another version is used for EIP712 domain separator in the contract constructor. We recommend utilizing only one version value (`__with_semver` here) for implementing the EIP712 meta-transaction standard.

L07. Missing events (fixed)

Consider emitting events on changing protocol parameters in the following functions:

- `setTotalClaimsLimits` and `setTierPerWalletLimits` of the **ERC721BaseSale** contract.
- `setTwapInterval` of the **UniswapV3Oracle** contract.

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

L08. Code quality: external functions (fixed)

Consider declaring the following functions as external instead of `public` when possible to improve code readability and optimize gas consumption.

- `executeBatch` and `getNonce` of the **CallForwarder**
- `getTradePrice` of the **UniswapV3Oracle** contract
- view functions of the **ERC721Properties** contract

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

L09. Code quality: assert instead of require (fixed)

In the `_initNextRole` function of the **App** contract at line 83 `assert` is used for handling errors. We recommend utilizing require here and using a constant for maximum number of roles.

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

L10. Redundant check (addressed)

`tokenSaleAddress_.code.length > 0` checks if an account has a non-empty code field. However, there is a call for `decimals()` on line 88, which would revert if the account had an empty code field at line 86 in **ERC721OpenSale**.

> *Such approach is chosen by developers to improve the maintaining and development process.*

L11. Redundant check (fixed)

The `getLeftClaimsForWallet` view function of the **ERC721BaseSale** contract checks (at line 153) if the number of claims exceeds the limit for a specific wallet. Since this condition is ensured while changing the value of `tierPerWalletClaimed` inside the `_claim` function, we recommend omitting the check in the getter since it should always pass.

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

L12. Redundant check (addressed)

The `_domainSeparatorV4` function of **EIP712** contract checks if it contract address is cached. As the caching and creating of the domain separator are performed during contract deployment, it is safe to return the cached value of the domain separator here.

> *Considered as not an issue by developers*

L13. Inconsistent initialization errors checking (fixed)

In the case of receiving tokens via the Merkle proof mechanism for `ERC721WhitelistSale` there are checks that the contract is properly initialized, i.e. Merkle root is set. However, similar `claim` functions within `ERC721WhitelistClaim` and `ERC721LinkClaim` do not perform such a check. Since the verification with a zero (uninitialized) Merkle root will not pass and the probability of incorrect deployment is small, we recommend removing the checks from **ERC721WhitelistSale**.

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

L14. Discrepancy with documentation (fixed)

Line 102 in the **CallForwarder** contract includes the description of using `_msgSender()` for retrieving the address from the signature. However, the code at line 105 does not utilize it, relying on `msg.sender`.

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

L15. Lack of documentation (addressed)

Functions of **ERC721Properties** have no documented description. We recommend covering the code with NatSpec comments, as it helps to avoid errors and accelerates the development process.

> *The developers have planned fixing this issue in the further update.*

Notes

N01. Weak random

The **ERC721Properties** contract allows using a pseudorandom mechanism for NFT minting. Note that this approach is vulnerable to pre-minting attack.

N02. Misleading comments

The comment for **ERC721BaseClaim** states that it implements the ERC721 standard. However, this contract of not follow the ERC721 interface itself, relying on **ERC721Properties**.

N03. Limited data in event

The `Deployed` event in the in `deploy` function of the **DAOConstructor** contract does not include information on calls for `_deployOpenSale()` and `_deployWhiteListSale()`. Consider adding the corresponding return values (unused in the current version) to the event.

N04. Function visibility

The `getAttribute` function of **ERC721Properties** executes only the call to the corresponding internal function. Consider removing the separation by utilizing the external function throughout the contract.

This analysis was performed by Pessimistic:

Vladimir Pomogalov, Security Engineer
Vladimir Tarasov, Security Engineer
Ivan Gladkikh, Junior Security Engineer
Irina Vikhareva, Project Manager

September 13, 2022