Ammbr Security Analysis

by Pessimistic

This report is private.

Published: May 21, 2021.

Abstract

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

The audit showed no critical issues. However, two medium severity and a number of low severity issues were found. They do not endanger project security. Nevertheless, we highly recommend addressing them.

General recommendations

We recommend adding `NatSpec` and public documentation to the project, utilizing development framework, adding tests, following the ERC20 standard, and addressing other issues mentioned in this report.

Project overview

Project description

For the audit, we were provided with Ammbr project on a private GitHub repository, commit 06d1c22717fac519b6a3e705e2220462eb87bb14.

The project has no documentation and no tests.

The project does not compile.

The total LOC of audited sources is 383.

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 tool SmartCheck.
    - We manually verify (reject or confirm) all found issues.
- 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 smart contracts operation in current implementation. We highly recommend addressing them.

ERC20 standard violation

EIP-20 states:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT
assume that false is never returned!
```

However, returned values from `transfer()` and `transferFrom()` calls are not checked in **ERC20Holder**.

Limited access

According to the user manual, minting process is decentralized and any user can mint NFT by calling **AmmbrNFT.createToken**. However, underlying code in **AmmbrERC721Master.mint** requires `msg.sender` to be explicitly white-listed by granting `MINTER` role by the contract owner.

Low severity issues

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

Code quality

- Consider declaring functions as `external` instead of `public` where possible in **ExchangeNFT**.
- There are typos in **ExchangeNFT** revert-messages at lines 62 and 69.
- Since pagination logic is not used on-chain, we recommend combining `getAsksByPage()` and `getAsksDescByPage()` functions into one with `from` and `to` parameters instead.
- Consider declaring `nft` and `quoteERC20` as `immutable` in **ExchangeNFT** contract, since they are set only during deployment. This also applies to `public ammbr` in **ERC20Holder**.
- In **ERC20Holder** contract, the name of `setFeeAddr()` function is confusing since it changes address of a token holder, not fee recipient.
- Consider in-lining `internal` functions `_deposit()` and `_withdraw()` in **ERC20Holder** contract since they are used only once.
- Since `msg.sender` is an `address`, explicit type casting is redundant in **ExchangeNFT** contract at lines 71 and 73.
- Consider including `i = 0` case in for-loops of **ExchangeNFT** contract and therefore updating `i > 0` condition with `i >= 0`.
- Consider removing `if-else` block and using only one for-cycle in `getAsksByPageDesc()` function of **ExchangeNFT** contract.
- Using an intermediate variable `msgSender` is unnecessary in **AmmbrNFT.createToken**.
- The check at line 49 of **ExchangeNFT.sol** is redundant, since the contract does not call itself and `msg.sender` cannot be zero address.
- **ERC20Holder.sol** contains misleading license identifier: the NFT license is not a valid SPDX license expression.

Gas consumption

- Math operations are safe by default since Solidity version `0.8.0`, so SafeMath library is redundant.
- Consider using local variable to store `_tokenSellers[_tokenId]` value in `buyToken()` function of **ExchangeNFT** contract to decrease gas consumption.
- The usage of `.length()` property in for-loop is expensive since it is read on each iteration of the cycle. Consider storing it to a local variable in **ExchangeNFT** contract to decrease gas consumption.
- Consider using `msg.sender` when removing seller address from `_userSellingTokens` in `cancelSellToken()` function of **ExchangeNFT** contract at line 82.

This analysis was performed by Pessimistic.

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

May 21, 2021