#HashEx
BLOCKCHAIN SECURITY

ApeDAO

smart contracts
preliminary audit report
for internal use only

September 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the ApeDAOFinance team to perform an audit of their smart contract. The audit was conducted between 17/09/2023 and 20/09/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @iotapes/liquid-apes Github repository after the fa2142f commit.

Update: the ApeDAOFinance team has responded to this report. The updated code is located in the @iotapes/liquid-apes GitHub repository after the 30fb681 commit.

Some of the contracts are designed to be deployed behind a proxy, meaning their implementations can be updated. Users must pay attention and do own research before using such contracts.

2.1 Summary

Project name	ApeDAO
URL	https://apedao.finance
Platform	Shimmer Network
Language	Solidity

2.2 Contracts

Name	Address
LilApeNFT	0x3F5ae5270b404fF94BB4d2f15A4f3b46f16470D1
OGApeNFT	0xf640ed4ADFD525a3DEae9FA76a840898d61009C1
PausableUpgradeable	
ApeDAOVaultUpgradeable	
ApeDAOMarket	0x6127C40AA99E5764Ebc730F8c3f7602E27EF07A3

3. Found issues

5
Total issues

●	High	1 (20%)
●	Medium	2 (40%)
●	Info	2 (40%)

C1e. ApeDAOVaultUpgradeable

ID	Severity	Title	Status
C1el84	● Medium	Fees limit	✓ Resolved
C1el83	● Medium	Insecure random	✓ Resolved
C1el85	● Info	Incorrect description	✓ Resolved

C1f. ApeDAOMarket

ID	Severity	Title	Status
C1fl87	● High	BASE_FEE coefficient	✓ Resolved
C1fl86	● Info	Lack of documentation	✓ Resolved

4. Contracts

C1b. LilApeNFT

Overview

An implementation of the ERC-721 NFT token standard built with OpenZeppelin templates. LilApeNFT token has a fixed total supply, and supports enumeration and royalties.

C1c. OGApeNFT

Overview

An implementation of the ERC-721 NFT token standard built with OpenZeppelin templates. OGApeNFT token has a fixed total supply, and supports enumeration and royalties.

C1d. PausableUpgradeable

Overview

A custom contract that inherits the Ownable authorization model from the OpenZeppelin library and allows setting a mapping of boolean flags for different types of pause statuses.

C1e. ApeDAOVaultUpgradeable

Overview

An implementation of the ERC-20 token standard built with OpenZeppelin templates. ApeDAOVaultUpgradeable can be minted in exchange for supported NFT tokens (LilApeNFT and OGApeNFT) with an updatable exchange rate. The contract is designed to be deployed behind a proxy, meaning its implementation can be updated. Users must pay attention and do their own research before using this contract.

Issues

C1el84 Fees limit ● Medium ✓ Resolved

The `setFees()` function is managed by the contract owner and allows to set new fee amounts. In the current implementation this function doesn't have any safety limits for input values. Thus, if the owner sets a commission of 100, then the user will not receive anything for the provided NFTs.

Recommendation

Consider adding limit for each type of the fee.

C1el83 Insecure random ● Medium ✓ Resolved

A random number for redeeming NFT can be computed with high probability in the `getRandomTokenIdFromVault()` function. There's no reputable way of obtaining the randomness on-chain, so the industry standard is using VRF oracles or gaining the seed from the project owners if users trust them.

Recommendation

Use oracles such as Chainlink VRF to get secure random results as soon as they are available on the network you are using.

Update

The updated code of `getRandomTokenIdFromVault()` function solves the problem of obtaining a random number by calling `iscSandbox.getEntropy()`. At the same time, such a contract can be attacked by another smart contract. The attacking contract may regularly try to buy a certain NFT at a discount as a random one and revert if it fails to get the desired NFT. To mitigate this risk, a check could be added to `getRandomTokenIdFromVault()` that `tx.origin == msg.sender`. And since the `redeem()` and `redeemTo()` functions are open for everyone, the whitelist functionality must be added too. Pseudo-code example:

```
function reeem() {
  if(!isAddressWhitelisted(msg.sender)) {
    require(tx.origin == msg.sender);
  }
  ...
}
```

Note that whitelisted contracts should then use the same `tx.origin == msg.sender` check.

But this approach may reduce the potential market of users. Before making any changes, we recommend a comprehensive assessment of the risks of reducing market coverage.

C1e85 Incorrect description ● Info ✓ Resolved

The `mint()`, `mintTo()` function allows to mint new ApeDAOVault (ERC20) tokens in exchange for the provided NFT.

But the description for these functions states that NFTs are minted.

We recommend correcting the description of these functions.

C1f. ApeDAOMarket

Overview

This contract allows users to buy and redeem ApeDAO NFTs using SMR tokens.

Issues

C1f87 BASE_FEE coefficient ● High ✓ Resolved

The `buyAndRedeemSMR()` function allows to buy NFTs for SMR tokens. To purchase NFTs, SMR tokens must first be exchanged for the DAOVault tokens.

To calculate the required amount of DAOVault tokens, a similar formula is used as in the `redeemTo()` function of the `ApeDAOVaultUpgradeable` contract.

But the `buyAndRedeemSMR()` function also applies an additional `BASE_FEE` coefficient. This significantly increases the number of DAOVault tokens that has to be bought. This also indirectly affects the value of `maxSMRIn`.

```
_buyVaultToken(auxAmount + (totalFee * BASE_FEE), maxSMRIn, path);
```

In this case, the user can satisfy the increased number of SMR tokens. But then only a part of the DAOVault tokens will be spent to buy NFTs in the ApeDAOVaultUpgradeable contract. The other part will be blocked in the ApeDAOMarket contract.

Recommendation

We recommend checking your fee calculations and testing this issue.

C1fl86 Lack of documentation ● Info ✓ Resolved

Consider adding documetation for the `SMR`, `apeDAOVault`, `uniswapRouter` state variables.

5. Conclusion

1 high, 2 medium severity issues were found during the audit. 1 high, 2 medium issues were resolved in the update.

The ApeDAOVaultUpgradeable contract is highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY