ExtractOre
Smart Contract Audit Report

IMMUNE BYTES

Audits

October 8, 2020

Introduction

1. About ExtractOre

Exgold is composed of 3 smart contracts:

- Exgold
- MinerCards
- MinerCardRewards

The project was developed using openzeppelin CLI, and buidler as the testing framework.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

Following document were provided by the client for the purpose of the audit:

- Read.me - File outlining the important details of ExtractOre

Audit Process & Methodology

ImmueBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract

Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: ExtractOre
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: 467970cb54ab18b046b10210f4b4a5d7668e9219

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped in the following three categories:

1. Security: Identifying security related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

IMMUNE BYTES

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	1	3	4
Closed			

High severity issues

1. **Funds can be drained from MinerCardRewards.sol.**
   In **MinerCardRewards.sol**, **setApprovals**() function gives allowance to the caller to use the contract's funds. Since this function is declared as public, it can be called by anyone with any amount as input. Any attacker can call this function with an infinite amount as input and can get approval to use all the funds held by contract. Then the attacker can drain all the different tokens held by **MinerCardRewards.sol**.

   Recommendation:
   Since implementation of this function is not required, consider removing this function from the smart contract. A contract must never give token allowance to the user/caller.

Medium severity issues

1. **No input validation for mint() and mintMultiple() function.**
   In **MinerCards.sol**, there is no check for the validity of **_id** parameter. The project only supports the minting of tokens with predefined ids, a check must be placed to ensure the id is valid. This function is protected by **onlyAdmin**() modifier and admin can be a smart

contract (MinerCardRewards) or a user's wallet, so invalid id can be provided intentionally or unintentionally by a human admin.

Recommendation:
Consider adding a check via **validateTokenType**() function in both the functions.

2. **Too much Admin controls in MinerCard.sol.**
   The **MinerCard** contract has too many admin rights which can be misused intentionally or unintentionally, i.e. in the case where the admin's private key gets compromised. As per the logic of smart contract there can be infinite admins for the **MinerCard** smart contract**.** Admin rights can be given to smart contracts or human users(EOA). All Admins have the equal right to mint any amount of tokens to any account. This can be a possible attack vector to the project, project's security will be compromised even if only one of those admin's private key gets compromised.

   Recommendation:
   Consider adding some governance logic for assigning admins or atleast use a multisig wallet as admin. Also if the admin rights are only meant to be given to MinerCardRewards contract, a single address variable can be used instead of **_admins** mapping. Also consider the need for **mintMultiple**() and **mintBatch**() functions as they are not used in the **MinerCardRewards** contract.

3. **No logic present to revoke admin rights.**
   According to the MinerCard contract implementation, admin rights can be given to any ethereum address, also there can be multiple admins for the contract. There should be a mechanism to revoke admin rights for an address but none is present. This could raise issues if any admin's private keys get compromised.

   Recommendation:
   Consider adding a function to revoke admin rights for an address.

Low severity issues

1. **Misleading comments.**
   The comments for **mint**() in **MinerCard.sol** and for **constructor**() in **MinerCardRewards.sol** are somewhat misleading. For **mint**() the comments don't specify the complete working of the function and also the all parameters are not explained. For constructor the comments are specific to upgradeable contracts which use **initializer**() as constructor. Either the contract should be upgradeable or comments are misleading.

Recommendation:
Consider correcting the comments for the above mentioned functions.

2. **Erc20 function calls should be wrapped in require.**
   As per the standard practices for erc20 tokens, the return value from their function calls should be checked and handled explicitly. In **MinerCardRewards.sol** the returned boolean value of Erc20 function calls at **line 128, 158 and 188** are not checked.

   Recommendation:
   Consider wrapping the above mentioned Erc20 function calls in **require** statements.

3. **No mechanism to deal with unforeseen bugs**
   There is no logic implemented in **MinerCard** and **MinerCardRewards** smart contracts to deal with future unforeseen bugs/upgrades.

   Recommendation:
   Consider adding a mechanism to pause the functioning of smart contracts if any vulnerability is detected in future. Ability to upgrade the code of smart contracts may also be considered.

4. **Functions should be declared as *external.***
   Most of the functions in **ExGold.sol**, **MinerCard.sol** and **MinerCardRewards.sol** are declared as public. Since some functions are never called internally and are only meant to be called externally they should be declared as **external**. External functions cost less gas than public functions.

   Recommendation:
   Consider declaring most of the functions as **external** instead of **public**.

Unit Test

All unit tests provided by ExGold are passing without any issues.

```
  Exgold
    ✓ should initialize contract with correct values (189ms)
    ✓ should call initialize function only once (110ms)
    ✓ should allow holder burn tokens (107ms)

  MinerCardRewards
    lockFunds(address, uint256, uint256)
TIME:  7776000
      ✓ should lock funds (249ms)
      ✓ should revert if invalid token ID (102ms)
      ✓ should revert if user has insufficient miner cards
      ✓ should revert if insufficient allowance set for MinerCardsRewards  contract (90ms)
      ✓ should revert if invalid lock amount provided (92ms)
    release()
      ✓ should revert if no funds locked (108ms)
      ✓ should revrt if account has no ERC-721 (152ms)
      ✓ should release locked funds (210ms)
    withdraw()
      ✓ should withdraw funds (230ms)
      ✓ should revert if account has no ERC-721 (155ms)
      ✓ should revert if current time is before release time (146ms)
      ✓ should revert if no funds locked
      ✓ should revert if insufficient funds to pay dividends (153ms)

  MinerCards
    mint(address, uint256, uint256)
      ✓ should mint 1 NFT with correct values (56ms)
      ✓ should revert when mint is called by unauthorized sender
      ✓ should revert when mint is called with a null destination address
    _mintBatch(address, uint256[] memory, uint256[] memory
      ✓ should mint tokens in batch (63ms)
      ✓ should update token supply (59ms)
      ✓ should revert when mintBatch is called with a null destination address
      ✓ should revert if length of inputs do not match
      ✓ should revert if mintBatch is called by unauthorized sender
      ✓ should revert when mintBatch is called with invalid token type
    safeTransferFrom(address, address, uint256, uint256, bytes calldata)
      ✓ should make a safeTransfer
    More...
      ✓ should add an admin
      ✓ should invalidate token


  28 passing (9s)
```

Coverage Report

Test coverage of ExGold's smart contract is **not** 100%.

```
-----------------------|----------|----------|----------|----------|----------------|
File                   |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------------|----------|----------|----------|----------|----------------|
 contracts/            |    92.16 |    86.84 |    91.18 |    92.38 |                |
  Exgold.sol           |      100 |      100 |      100 |      100 |                |
  MinerCardRewards.sol |       90 |    86.67 |     87.5 |    90.14 |... 228,229,261 |
  MinerCards.sol       |    96.15 |     87.5 |    93.75 |    96.43 |            133 |
-----------------------|----------|----------|----------|----------|----------------|
All files              |    92.16 |    86.84 |    91.18 |    92.38 |                |
-----------------------|----------|----------|----------|----------|----------------|
```

Recommendation:
We recommend 100% line and branch coverage for unit test cases.

Slither Tool Result

```
MinerCardRewards.lockFunds(uint256,uint256) (contracts/MinerCardRewards.sol#99-141) ignores return value by
token.transferFrom(_account,address(this),_lockAmount) (contracts/MinerCardRewards.sol#128)
```

```
lockFunds(uint256,uint256) should be declared external:
        - MinerCardRewards.lockFunds(uint256,uint256) (contracts/MinerCardRewards.sol#99-141)
```

Notes

1. ***How it Works*** section of **Readme.md** suggests to send some ETH to **MinerCardRewards** contract for minting and transferring of tokens while practically this is not required as well as is of no use. The cost of minting and transferring of tokens will be borne by the caller of the transaction (user/admin) always.

Concluding Remarks

While conducting the audits of ExtractOre's smart contract, it was observed that the contracts contain High, Medium, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by the ExtractOre's developers. Resolving the areas of recommendations are up to ExtractOre's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse ExtractOre's platform or it's the product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the one audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.