Ekta

Swap Contract

Smart Contract Audit Final Report

March 15, 2022

This 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.

Introduction

1. About Ekta

Ekta's vision is to create a world where blockchain technology is used to give everyone a chance to live a better life. A new ecosystem is needed, one where people from different backgrounds and socio-economic circumstances can participate freely, without the barriers and inefficiencies introduced by centralized governing bodies.

Ekta's mission is to bridge the blockchain world with the world we live in and to create value in both. This is accomplished through various branches of the Ekta ecosystem, which include:

- The tokenization of real-world assets through Ekta Chain and Ekta's self-developed NFT Marketplace
- Ekta's decentralized credit platform that allows all users to participate
- Physical spaces such as the island chain currently being developed in Indonesia, where physical land and real estate assets will be brought on-chain
- Ekta's startup incubator and innovation center open to retail investment

Visit https://ekta.io/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the 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 145+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups 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

The Ekta team has provided the following doc for the purpose of audit:

1. https://whitepaper.ektaworld.io/

This 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.

Audit Process & Methodology

ImmuneBytes 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: Ekta
- Contracts Name: Swap.sol
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: fd06b66292a4d4ed5ad81d364d3a2fbe0c553313
- Github commit/Smart Contract Address for audit:: 105053b820fd5fc7030c13cbacd8ac5a90007c62
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

This 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.

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 into 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

Security Level Reference

Every issue in this report were 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	-	-	-
Closed	-	-	2

This 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.

Contract: Swap.sol

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **Return Value of a function call is not used effectively.**
   Line no: 34, 35

   Explanation:
   The functional calls **transfer** and **transferFrom** made in the above-mentioned lines do return a boolean value that indicates whether or not the call made was successful.

   These boolean return values can be used in the function as a check to ensure that the execution of the function is only allowed if the function update was successfully made.

   Recommendation:
   A require statement can be used to check and ensure that the function call was done successfully or **safeTransfer** and **safeTransferFrom** can be used alternatively.

   Amended (March 15th, 2022): The issue was fixed by the **Ekta** team and is no longer present in commit 105053b820fd5fc7030c13cbacd8ac5a90007c62.

2. **Absence of Event Emission after imperative State variable Update**

   Explanation:
   Functions that update an imperative arithmetic state variable contract should emit an event after the update.

   As per the current architecture of the contract, the following function update crucial state variable but doesn't emit any event on its modification:
   - **updateSwapToken()**

This 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.

The absence of event emission for important state variables update also makes it difficult to track them off-chain as well.

Recommendation:
As per the best practices in smart contract development, an event should be fired after changing crucial arithmetic state variables.

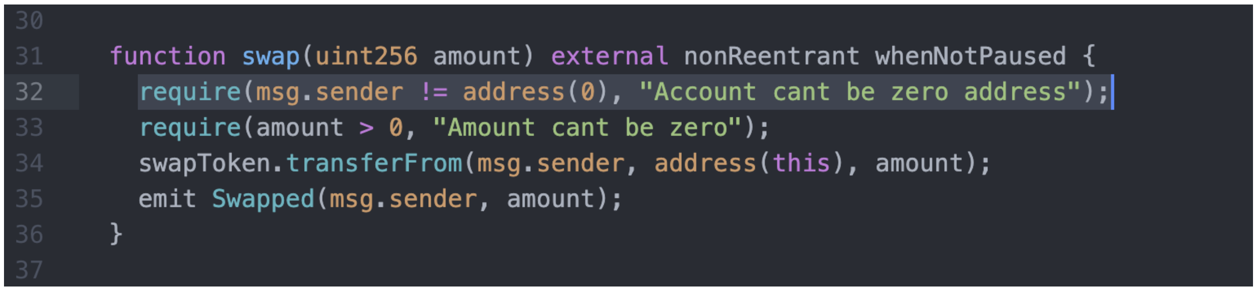
Amended (March 15th, 2022): The issue was fixed by the **Ekta** team and is no longer present in commit 105053b820fd5fc7030c13cbacd8ac5a90007c62.

Recommendations / Informational

1. **Redundant require statement found in the swap() function**
   Line no: 32

   Explanation:
   The **swap()** function in the contract, includes a require statement that checks that the caller of the function must not be a **zero address**.

   However, the **msg.sender** of a function can never be a zero address since the private key of an absolute zero address is not owned by anyone and is probably non-existing.



```
30
31    function swap(uint256 amount) external nonReentrant whenNotPaused {
32      require(msg.sender != address(0), "Account cant be zero address");
33      require(amount > 0, "Amount cant be zero");
34      swapToken.transferFrom(msg.sender, address(this), amount);
35      emit Swapped(msg.sender, amount);
36    }
37
```

   Recommendation:
   require statements in solidity are primarily useful for validating the inputs passed to a function. Since there aren't any address arguments being passed in this function at the moment, the **require** statement can be removed from the function. This shall effectively help in the gas optimization of the function as well.

   Amended (March 15th, 2022): The issue was fixed by the **Ekta** team and is no longer present in commit 105053b820fd5fc7030c13cbacd8ac5a90007c62.

This 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.

2. **NatSpec Annotations must be included**

   Explanation:
   The smart contracts do not include the NatSpec annotations adequately.

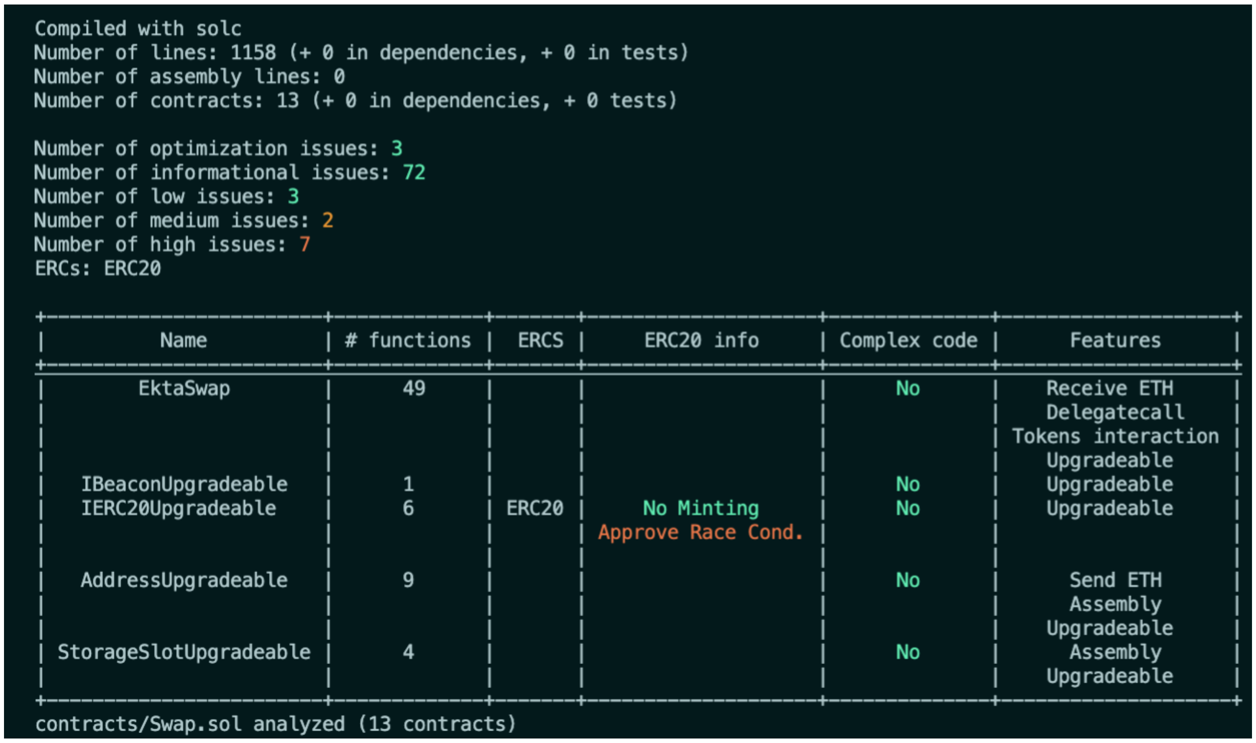
   Recommendation:
   Cover by NatSpec all Contract methods.

   Amended (March 15th, 2022): The issue was fixed by the **Ekta** team and is no longer present in commit 105053b820fd5fc7030c13cbacd8ac5a90007c62.

This 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.

Automated Audit Result

1. Swap.sol



```
Compiled with solc
Number of lines: 1158 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 72
Number of low issues: 3
Number of medium issues: 2
Number of high issues: 7
ERCs: ERC20

+-------------------------+-------------+------+--------------------+--------------+-------------------+
|          Name           | # functions | ERCS |     ERC20 info     | Complex code |     Features      |
+-------------------------+-------------+------+--------------------+--------------+-------------------+
|        EktaSwap         |     49      |      |                    |      No      |    Receive ETH    |
|                         |             |      |                    |              |    Delegatecall   |
|                         |             |      |                    |              | Tokens interaction |
|                         |             |      |                    |              |    Upgradeable    |
|   IBeaconUpgradeable    |      1      |      |                    |      No      |    Upgradeable    |
|   IERC20Upgradeable     |      6      | ERC20 |     No Minting    |      No      |    Upgradeable    |
|                         |             |      | Approve Race Cond. |              |                   |
|                         |             |      |                    |              |                   |
|   AddressUpgradeable    |      9      |      |                    |      No      |     Send ETH      |
|                         |             |      |                    |              |     Assembly      |
|                         |             |      |                    |              |    Upgradeable    |
| StorageSlotUpgradeable  |      4      |      |                    |      No      |     Assembly      |
|                         |             |      |                    |              |    Upgradeable    |
+-------------------------+-------------+------+--------------------+--------------+-------------------+
contracts/Swap.sol analyzed (13 contracts)
```

Concluding Remarks

While conducting the audits of the Ekta smart contracts, it was observed that the contracts contain only Low severity issues.

Our auditors suggest that Low severity issues should be resolved by the developers. 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 the Ekta platform or its product nor this audit is investment advice.
Notes:

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

ImmuneBytes