Ekta World

BridgeEthDev, BridgeBscDev, EktaManagerDev

Smart Contract Audit Report

December 24, 2021

Introduction

1. About Ekta World

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://ektaworld.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 105+ 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

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

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

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 World
- Contracts Name: BridgeEthDev, BridgeBscDev, EktaManagerDev
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: Null
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

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 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	1	4
Closed	-	-	-

Contract: BridgeEthDev & BridgeBscDev

High Severity Issues

1. **The contract restricts the entry of ETH/BNB**
   Line no: 169-175

   Explanation:
   The **Bridge contracts** include some specific functions such as **withdrawEthFromContract** and **withdrawBnbFromContract**.

   These functions allow the admin to withdraw ETH or BNB from the contract to a particular user address passed in the function arguments.

```
168
169   function withdrawEthFromContract(address user, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
170     require(user != address(0), "Bridge: Address cant be zero address");
171     require(amount <= address(this).balance, "Bridge: Amount exceeds balance");
172     address payable _user = payable(user);
173     _user.transfer(amount);
174     emit EthFromContractTransferred(user, amount);
175   }
176
```

   However, the contracts never include an adequate path to allow the entry of ETH/BNB into it. This is because the protocol neither includes any **payable fallback** function nor has a function with the **payable keyword** attached to it.

   This leads to an undesirable scenario where any amount of ETH/BNB sent to the contract shall be reverted back and never be stored in the contract although the contract expects it as it includes a withdraw() function specifically to transfer ETH/BNB from the contract to an address.

   Furthermore, also affects the significance of the **withdrawETHFromContract()/ withdrawBnbFromContract** functions in the contract as they can never withdraw any amount since the contract will never be able to receive any ETH or BNB in the first place.

   Recommendation:
   If the above-mentioned contracts are supposed to receive ETH or BNB, proper **payable** or **fallback** functions must be added to the contract to ensure that the protocol is capable of receiving ether.

   However, if the contract is not supposed to deal with ETH or BNB at all, the above-mentioned withdrawal functions shall be removed.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **Inadequate checkpoints for withdrawERC20Token() function.**
   Line no: 162-167

   Explanation:
   The **withdrawERC20Token()** function allows the admin to pass in any token address that exists in the contract and initiate a transfer for the token from the contract to the admin address.

```
161
162   function withdrawERC20Token(address _tokenContract, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
163     require(_tokenContract != address(0), "Bridge: Address cant be zero address");
164     IERC20 tokenContract = IERC20(_tokenContract);
165     tokenContract.transfer(msg.sender, amount);
166     emit TokenFromContractTransferred(_tokenContract, msg.sender, amount);
167   }
```

   However, before the transfer of tokens is initiated, it is not ensured whether or not the contract has the required amount of token balance for the specific token address passed as an argument.
   While this doesn't break the expected behavior of the contract, it's an imperative filter to ensure only valid token addresses with sufficient balance in the contract shall be allowed to enter the function.

   Recommendation:
   A **require** statement with adequate error messages could be included to ensure that the token addresses being passed as arguments are not invalid.

2. **Redundant comparisons to boolean Constants**
   Line no: 100, 128

   Explanation:
   Boolean constants can directly be used in conditional statements or require statements. Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require** statements.

```
125   function withdrawNativeToToken(address user, uint256 amount, uint256 nonce, bytes calldata signature)
126     require(amount > 0, "Bridge: Amount cant be zero or negative numbers");
127     // check for nonce
128     require(processedNonces[user][nonce] == false, 'Bridge: Transfer already processed');
129     // verify signature
```

   Recommendation:
   The equality to boolean constants could be removed from the above-mentioned line.

3. **Absence of Zero Address Validation**
   Line no- 37-44

   Explanation:
   The **BridgeEthDev** and **BridgeBscDev** contract includes a constructor that updates some of the imperative addresses in the contract like **admin**, **tokenToSwapWithNative**.

   However, during the automated testing of the contract, it was found that no Zero Address validation has been implemented on addresses passed as an argument for these state variables.

   Recommendation:
   A **require** statement should be included in such functions to ensure no zero addresses are passed in the constructor arguments.

Recommendations / Informational

1. **Invalid function name found in BridgeBscDev contract**
   Line no: 162

   Explanation:
   The BridgeBscDev contract includes a function with the name **withdrawERC20Token()**.

```
161
162   function withdrawERC20Token(address _tokenContract, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
163     require(_tokenContract != address(0), "Bridge: Address cant be zero address");
164     IERC20 tokenContract = IERC20(_tokenContract);
165     tokenContract.transfer(msg.sender, amount);
166     emit TokenFromContractTransferred(_tokenContract, msg.sender, amount);
167   }
```

   This is an invalid function name as ERC20 tokens standards are available only on the Ethereum blockchain and not on Binance smart chain

   Recommendation:
   Functions names should be assigned adequately.

2. **NatSpec Annotations must be included**

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

   Recommendation:
   Cover by NatSpec all Contract methods.

3. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Explanation:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.
   It's not considered a better practice in Smart contract development to do so as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

Contract: EktaManagerDev

High Severity Issues

No issues were found.

Medium Severity Issues

1. **Transfer of ETH might fail due to Gas Constraints**
   Line no: 114

   Explanation:
   The **withdrawEthFromContract()** function allows the admin to withdraw ETH from the contract to any specific user or contract address.

```
110     function withdrawEthFromContract(address user, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
111         require(user != address(0), "Bridge: Address cant be zero address");
112         require(amount <= getBalance(), "Bridge: Amount exceeds balance");
113         address payable _user = payable(user);
114         _user.transfer(amount);
115         emit EthFromContractTransferred(user, amount);
116     }
117
```

   Additionally, the function uses the **.transfer()** method to initiate a transfer of ether from the contract to the address.

   However, if the recipient of the ether is a smart contract, the transfer might fail. This is because methods like **transfer() & send()** when used for sending ether, always forward a fixed amount of gas, i.e., **2300**.

   If the recipient smart contract uses more than this amount of gas to execute any further transaction, the transfer of ether will never succeed in such a scenario.

   Recommendation:
   It is recommended to use the **call()** method for initiating ether transfers from the contract instead of **transfer()** or **send()**. For more details, refer to this article.

Low Severity Issues

1. **Absence of Zero Address Validation**
   Line no- 27-33

   Explanation:
   The **EktaManagerDev** contract includes a constructor that updates imperative addresses in the contract like the **owner's** address.
   However, during the automated testing of the contract, it was found that no Zero Address validation has been implemented on addresses passed as an argument for these state variables.

   Recommendation:
   A **require** statement should be included in such functions to ensure no zero addresses are passed in the constructor arguments.

Recommendations / Informational

1. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Explanation:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.

   It's not considered a better practice in Smart contract development to do so as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

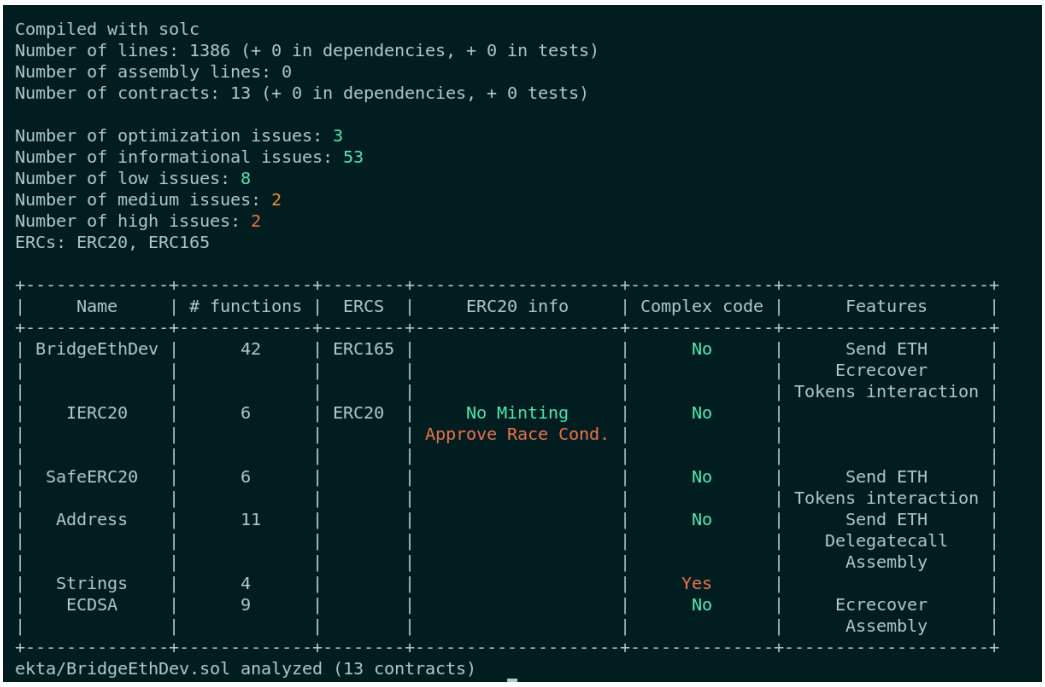
2. **NatSpec Annotations must be included**

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

   Recommendation:
   Cover by NatSpec all Contract methods.

Automated Audit Result

1. BridgeEthDev

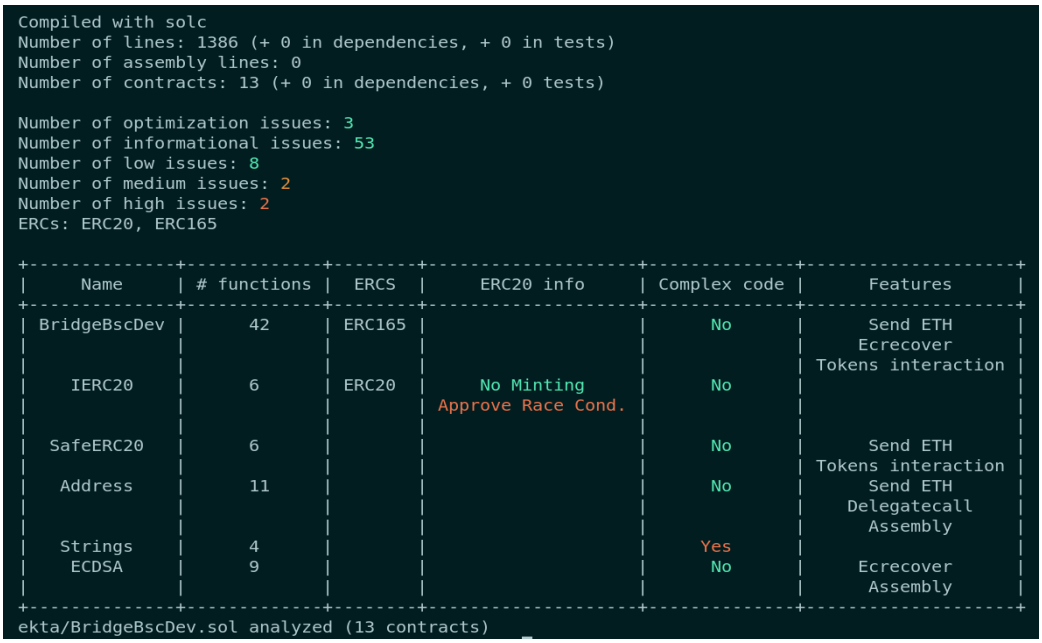


```
Compiled with solc
Number of lines: 1386 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 53
Number of low issues: 8
Number of medium issues: 2
Number of high issues: 2
ERCs: ERC20, ERC165

+--------------+-------------+--------+--------------------+--------------+--------------------+
|     Name     | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+--------------+-------------+--------+--------------------+--------------+--------------------+
| BridgeEthDev |      42     | ERC165 |                    |      No      |      Send ETH      |
|              |             |        |                    |              |     Ecrecover      |
|              |             |        |                    |              | Tokens interaction |
|    IERC20    |      6      | ERC20  |     No Minting     |      No      |                    |
|              |             |        | Approve Race Cond. |              |                    |
|              |             |        |                    |              |                    |
|  SafeERC20   |      6      |        |                    |      No      |      Send ETH      |
|              |             |        |                    |              | Tokens interaction |
|   Address    |      11     |        |                    |      No      |      Send ETH      |
|              |             |        |                    |              |    Delegatecall    |
|              |             |        |                    |              |      Assembly      |
|   Strings    |      4      |        |                    |     Yes      |                    |
|    ECDSA     |      9      |        |                    |      No      |     Ecrecover      |
|              |             |        |                    |              |      Assembly      |
+--------------+-------------+--------+--------------------+--------------+--------------------+
ekta/BridgeEthDev.sol analyzed (13 contracts)
```

2. BridgeBscDev



```
Compiled with solc
Number of lines: 1386 (+ 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: 53
Number of low issues: 8
Number of medium issues: 2
Number of high issues: 2
ERCs: ERC20, ERC165

+--------------+-------------+--------+--------------------+--------------+--------------------+
|     Name     | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+--------------+-------------+--------+--------------------+--------------+--------------------+
| BridgeBscDev |      42     | ERC165 |                    |      No      |      Send ETH      |
|              |             |        |                    |              |     Ecrecover      |
|              |             |        |                    |              | Tokens interaction |
|    IERC20    |      6      | ERC20  |     No Minting     |      No      |                    |
|              |             |        | Approve Race Cond. |              |                    |
|              |             |        |                    |              |                    |
|  SafeERC20   |      6      |        |                    |      No      |      Send ETH      |
|              |             |        |                    |              | Tokens interaction |
|   Address    |      11     |        |                    |      No      |      Send ETH      |
|              |             |        |                    |              |    Delegatecall    |
|              |             |        |                    |              |      Assembly      |
|   Strings    |      4      |        |                    |     Yes      |                    |
|    ECDSA     |      9      |        |                    |      No      |     Ecrecover      |
|              |             |        |                    |              |      Assembly      |
+--------------+-------------+--------+--------------------+--------------+--------------------+
ekta/BridgeBscDev.sol analyzed (13 contracts)
```

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.

3. EktaManagerDev

```
Compiled with solc
Number of lines: 804 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 11 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 29
Number of low issues: 4
Number of medium issues: 0
Number of high issues: 3
ERCs: ERC20, ERC165

+----------------+-------------+--------+--------------------+--------------+--------------------+
|      Name      | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+----------------+-------------+--------+--------------------+--------------+--------------------+
|   IedgeERC20   |      2      |        |                    |      No      |                    |
| EKTAManagerDev |      41     | ERC165 |                    |      No      |    Receive ETH     |
|                |             |        |                    |              |      Send ETH      |
|                |             |        |                    |              | Tokens interaction |
|     IERC20     |      6      | ERC20  |     No Minting     |      No      |                    |
|                |             |        | Approve Race Cond. |              |                    |
|                |             |        |                    |              |                    |
|    Strings     |      4      |        |                    |     Yes      |                    |
+----------------+-------------+--------+--------------------+--------------+--------------------+
ekta/EKTAMannagerDev.sol analyzed (11 contracts)
```

Unit Test

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: EktaManager
    Add Token Pairs
      User with mapper role adding token pairs
        ✓ Should pass when user with mapper role adds token pairs (44ms)
        ✓ Should fail when user adds token pair for already existing root token (348ms)
      User with no mapper role adding token pairs
        ✓ Should fail when user with no mapper role adds token pairs (151ms)
      Zero address handling
        ✓ Should fail when user adds zero address as root token
        ✓ Should fail when user adds zero address as child token (51ms)
        ✓ Should fail when user adds zero address as root and child token (41ms)
    Update Token Pairs
      User with mapper role updates token pairs
        ✓ Should fail when user updates token pair for existing pair (45ms)
        ✓ Should fail when user updated token pair for non-existing pair
        ✓ Should pass when user with mapper role updates token pairs
      User with no mapper role updating token pairs
        ✓ Should fail when user with no mapper role updates token pairs (210ms)
      Zero address handling
        ✓ Should fail when user updates zero address as root token
        ✓ Should fail when user updates zero address as child token
        ✓ Should fail when user updates zero address as root and child token (43ms)
    Deposit
      Deposit amount to bridge
        ✓ Should fail when user deposits amount to bridge with insufficient allowances (53ms)
        ✓ Should pass when user deposits amount to bridge with sufficient allowances (87ms)
      Handle Zero address and zero amount
        ✓ Should fail when user deposits zero amount (54ms)
    Withdraw
      Withdraw token on ekta
        ✓ Should pass when user withdraws the token deposited (42ms)
        ✓ Should fail when user withdraws the token greater than the deposited value (99ms)
    Clean Token Pairs
      User with mapper role cleans token pairs
        ✓ Should pass when user with mapper role cleans token pairs
      User with no mapper role cleans token pairs
        ✓ Should fail when user with no mapper role adds token pairs (138ms)
    Pause/UnPause
      Pause/UnPause by user with Pauser role
        ✓ Should pass when user with Pauser role pauses the contract
        ✓ Should pass when user with Pauser role unpauses the contract
      Pause/UnPause by user with no Pauser role
        ✓ Should fail when user with no Pauser role pauses the contract
        ✓ Should fail when user with no Pauser role unpauses the contract (39ms)


  24 passing (8s)
```

```
Contract: BRIDGE
  Initial State
    ✓ has the correct name
  Add Token Pairs
    User with mapper role adding token pairs
      ✓ Should pass when user with mapper role adds token pairs (43ms)
      ✓ Should fail when user adds token pair for already existing root token (433ms)
    User with no mapper role adding token pairs
      ✓ Should fail when user with no mapper role adds token pairs (138ms)
    Zero address handling
      ✓ Should fail when user adds zero address as root token (38ms)
      ✓ Should fail when user adds zero address as child token (55ms)
      ✓ Should fail when user adds zero address as root and child token (52ms)
  Update Token Pairs
    User with mapper role updates token pairs
      ✓ Should fail when user updates token pair for existing pair (49ms)
      ✓ Should fail when user updated token pair for non-existing pair
      ✓ Should pass when user with mapper role updates token pairs
    User with no mapper role updating token pairs
      ✓ Should fail when user with no mapper role updates token pairs (135ms)
    Zero address handling
      ✓ Should fail when user updates zero address as root token (40ms)
      ✓ Should fail when user updates zero address as child token
      ✓ Should fail when user updates zero address as root and child token (57ms)
  Deposit
    Deposit amount to bridge
      ✓ Should fail when user deposits amount to bridge with insufficient allowances (82ms)
      ✓ Should pass when user deposits amount to bridge with sufficient allowances (160ms)
    Handle Zero address and zero amount
      ✓ Should fail when user gives zero address as root Token
      ✓ Should fail when user deposits zero amount (53ms)
  Clean Token Pairs
    User with mapper role cleans token pairs
      ✓ Should pass when user with mapper role cleans token pairs (38ms)
    User with no mapper role cleans token pairs
      ✓ Should fail when user with no mapper role adds token pairs (181ms)
  Pause/UnPause
    Pause/UnPause by user with Pauser role
      ✓ Should pass when user with Pauser role pauses the contract
      ✓ Should pass when user with Pauser role unpauses the contract
    Pause/UnPause by user with no Pauser role
      ✓ Should fail when user with no Pauser role pauses the contract
      ✓ Should fail when user with no Pauser role unpauses the contract


24 passing (9s)
```

Concluding Remarks

While conducting the audits of the Ekta World smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues.

Our auditors suggest that High, Medium, and 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 World 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