Bird Money

Smart Contract Audit Report

IMMUNE BYTES

Audits

Bird.Money

April 13, 2021

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. Abou BIrd.Money

Bird Money is combining bleeding-edge technologies to deliver new DeFi products to all members of our flock.
The mission is to harness the power of data to create a financial ecosystem tailored to you. We're using our off-chain oracle analytics to provide protocols with trust data that's available on-chain.

Visit to know more about https://bird.money/

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 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

Bird Money team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://docs.bird.money/

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: Bird Money
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: a7bcb6490875daaf8c414d9189f40254ec489435

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	2	4	6
Closed	-	-	-

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.

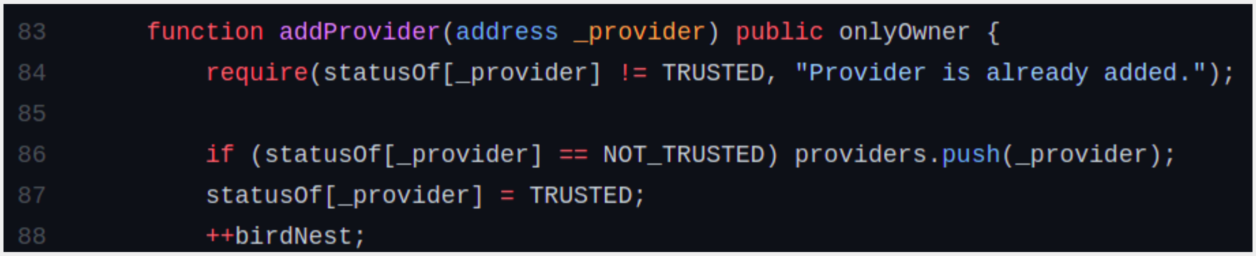
High severity issues

1. **addProvider function allows passing an address with WAS_TRUSTED status as an argument**

   Line no: 83-91
   Description:
   The **addProvider** function is designed in a way that only allows the addresses with the status **NOT_TRUSTED** to be added as a Trusted provider address.

   However, as per the current design of the function, an address with the status **WAS_TRUSTED** can be added as a trusted provider address as well. In other words, an address that has been removed from the trusted providers list(using **removeProvider** function) can once again be included as a trusted address.

   IS THIS INTENDED?



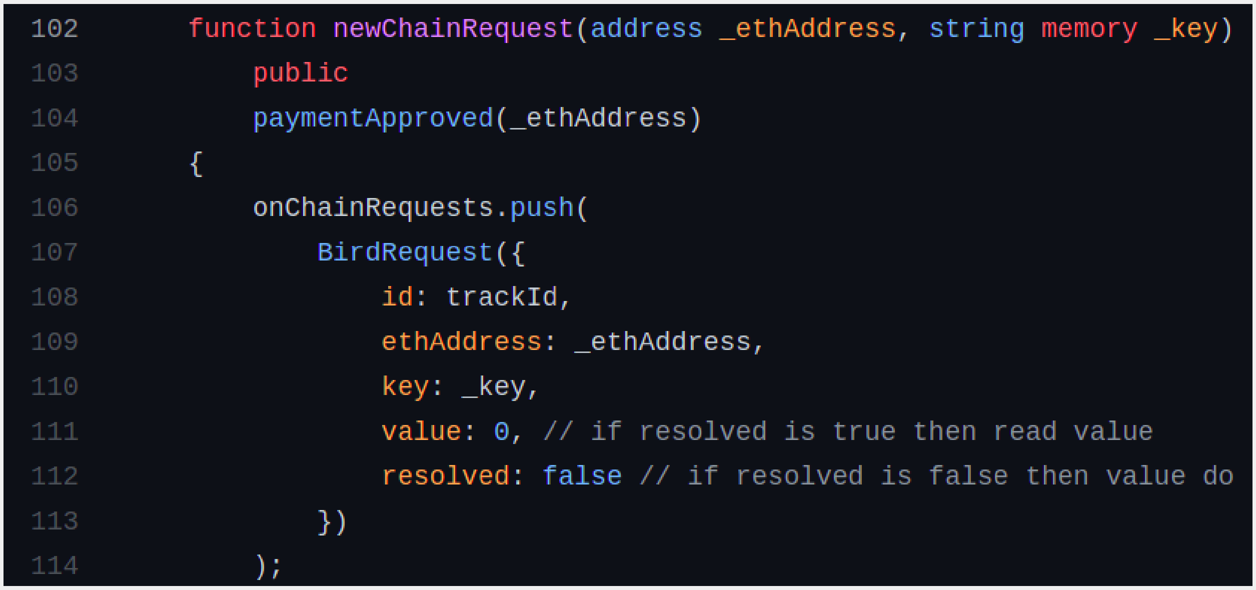
```
83      function addProvider(address _provider) public onlyOwner {
84          require(statusOf[_provider] != TRUSTED, "Provider is already added.");
85
86          if (statusOf[_provider] == NOT_TRUSTED) providers.push(_provider);
87          statusOf[_provider] = TRUSTED;
88          ++birdNest;
```

   Recommendation:
   If the above-mentioned scenario is not intended, then the function should be updated to handle such cases.

   If an address with a status **WAS_TRUSTED** is not supposed to be counted as a trusted provider again, then the addProvider function must include the following **require** statement within the function body:

   require(statusOf[_provider] != WAS_TRUSTED, "Provider cannot be added again");

2. **newChainRequest function allows Empty Strings to be passed as arguments**
   Line no: 102-121
   Description:
   The **newChainRequest** function doesn’t validate the string input passed to it and hence allows empty strings(_key) to be passed as an argument.



```
102      function newChainRequest(address _ethAddress, string memory _key)
103          public
104          paymentApproved(_ethAddress)
105      {
106          onChainRequests.push(
107              BirdRequest({
108                  id: trackId,
109                  ethAddress: _ethAddress,
110                  key: _key,
111                  value: 0, // if resolved is true then read value
112                  resolved: false // if resolved is false then value do
113              })
114          );
```

   If the **key** plays a significant role in the request, it might be difficult to track if empty strings are passed as **keys**.

   Recommendation:
   In order to avoid this scenario, the function must include the following **require** statement to ensure that an empty string is not passed as an address:

   require (bytes(_key).length>0,"String with ZERO length not allowed");

Medium severity issues

1. **Loops are extremely costly**
   Line no - 169, 217
   Description:
   The **for loops** in the **BirdOracle** contract include state variables like **.length** of a non-memory array in the condition of the for loops.
   As a result, these state variables consume a lot more extra gas for every iteration of the loop.

The following functions include such loops at the mentioned lines:

- **getProviders()** at Line 169
- **rewardProviders()** at Line 217

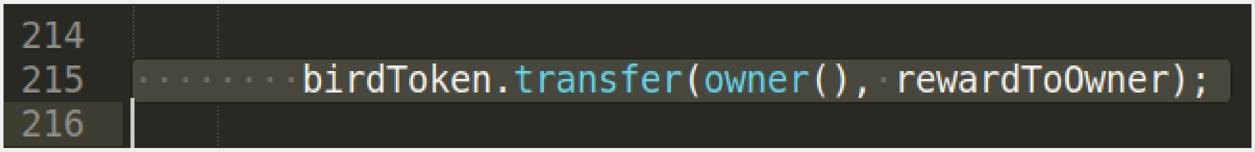
Recommendation:
It’s quite effective to use a local variable instead of a state variable like **.length** in a loop.

For instance,

```
local_variable = providers.length
for (uint i = 0; i < local_variable; i ++) {
        if (statusOf[providers[i]] == TRUSTED) {
            trustedProviders[t_i] = providers[i];
            t_i++;
        }
}
```

2. **Return Value of an External Call is Not used Effectively**

Line no - 186, 215, 219
Explanation:
The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.
These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.
However, the BirdOracle contract never uses these return values throughout the contract.



```
214
215         birdToken.transfer(owner(), rewardToOwner);
216
```

Recommendation:
Effective use of all the return values from external calls must be ensured within the contract.

3. setMinConsensus function doesn’t emit any event

Line no - 232

Description:

The **setMinConsensus** function modifies the state of a very crucial arithmetic state variable, i.e. **minConsensus**, but doesn’t emit any event after the updation of those variables.
Since there is no event emitted on updating this variable, it might be difficult to track it off-chain.

Recommendation:

An event should be fired after changing the imperative arithmetic variables.

4. State Variables updated after External Call

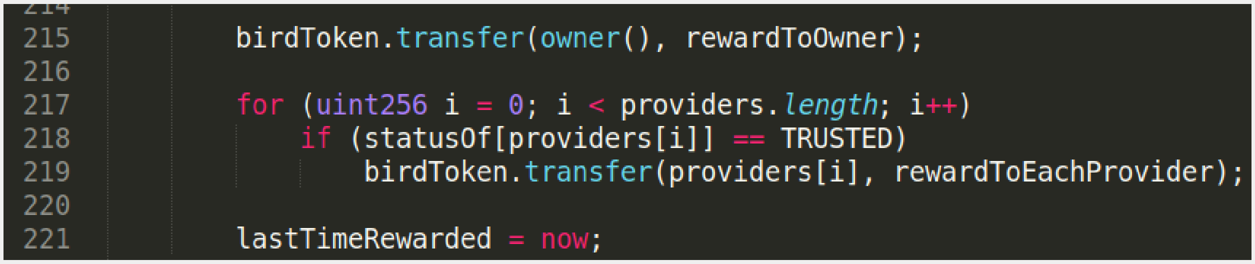
Line - 184-195, 199-222

Description:

The **BirdOracle** contract includes some functions that modify the state variables of the contract after making external calls.

- The **sendPayment** function makes an external call at Line 186 but updates the state of the **dueDateOf** mapping at Line 191 & 193, after the external call was made.
- Similarly, The **rewardProviders** function makes 2 external call at Line 215 and 219. However, it updates the state of the **lastTimeRewarded** state variable at Line 221, after the external call was made.

Although these external calls are made to the **Bird Token** contract itself, it is not considered a better practice in Solidity to update State Variables after making an External call. This violates the Check Effects Interaction Pattern.



```
214
215         birdToken.transfer(owner(), rewardToOwner);
216
217         for (uint256 i = 0; i < providers.length; i++)
218             if (statusOf[providers[i]] == TRUSTED)
219                 birdToken.transfer(providers[i], rewardToEachProvider);
220
221         lastTimeRewarded = now;
```

Recommendation:

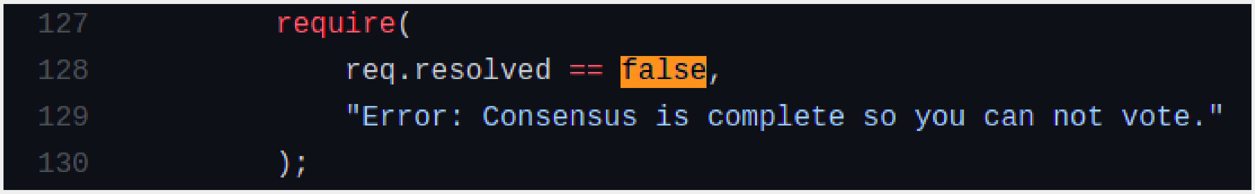
The Check Effects Interaction Pattern must be followed and State Variables should be updated before making any external call to another contract.

Low severity issues

1. **External Visibility should be preferred**
   Explanation:
   Those functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.

   Therefore, the following function must be marked as **external** within the contract:
   - **addProvider**
   - **removeProvider**
   - **newChainRequest**
   - **updatedChainRequest**
   - **getRatingByAddress**
   - **getRating**
   - **getProviders**
   - **sendPayment**
   - **rewardProviders**
   - **isApproved** at Line 224
   - **setMinConsensus**

2. **Comparison to boolean Constant**
   Line no: 128
   Description:
   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.



```
127            require(
128                req.resolved == false,
129                "Error: Consensus is complete so you can not vote."
130            );
```

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

3. Functions with similar names should be avoided

Line no - 23 and 52, 224 and 228

Description:
The BirdOracle contract includes a few functions with exactly similar names.
Since every function has different behavior, it is considered a better practice to avoid similar names for 2 different functions to eliminate any confusion and enhance the readability of the code.

Mentioned below are the functions with similar names but different behavior and arguments:

- **statusOf** at Line 23
- **statusOf** at Line 52
- **isApproved**() at Line 224
- **isApproved**() at Line 228

Recommended:
It is recommended to avoid using a similar name for different functions.

4. No visibility keyword assigned to statusOf Mapping

Line no: 23
Description:
The **statusOf** mapping has not been assigned any visibility keyword. If no visibility is assigned explicitly, the mapping will be assigned the default visibility. This might lead to unwanted difficulties while accessing this mapping from outside the contract.
Recommendation:
The mapping should be assigned a visibility keyword.

5. Order of layout

Description:
The order of functions as well as the rest of the code layout does not follow the solidity style guide.
Layout contract elements in the following order:
a. Pragma statements
b. Import statements
c. Interfaces
d. Libraries
e. Contracts
Inside each contract, library or interface, use the following order:

a. Type declarations
b. State variables
c. Events
d. Functions
Please read the following documentation links to understand the correct order: -
https://docs.soliditylang.org/en/v0.6.12/style-guide.html#order-of-layout
https://docs.soliditylang.org/en/v0.6.12/style-guide.html#order-of-functions

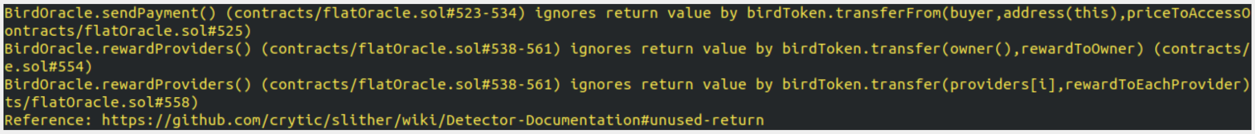
6. NatSpec Annotations must be included

Description:
Smart contract does not include the NatSpec annotations adequately.

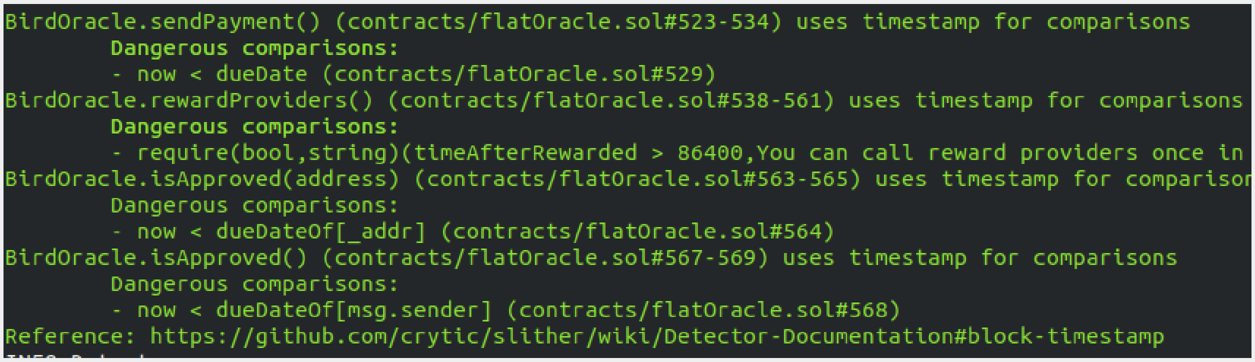
Recommendation:
Cover by NatSpec all Contract methods.

Automated Audit

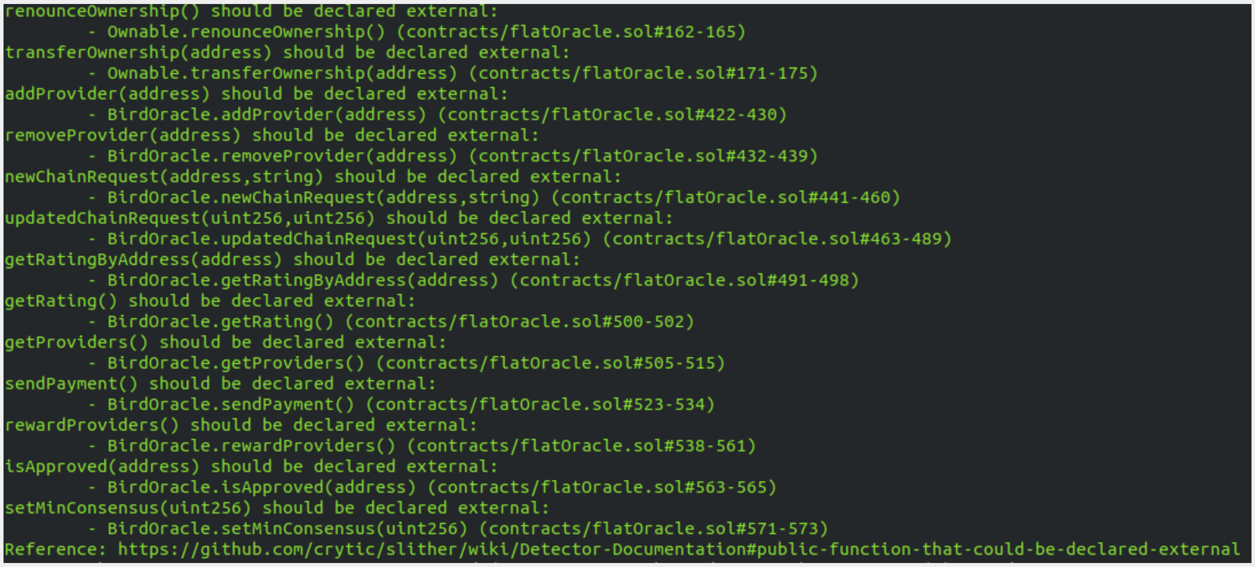
Slither



```
BirdOracle.sendPayment() (contracts/flatOracle.sol#523-534) ignores return value by birdToken.transferFrom(buyer,address(this),priceToAccessO
ontracts/flatOracle.sol#525)
BirdOracle.rewardProviders() (contracts/flatOracle.sol#538-561) ignores return value by birdToken.transfer(owner(),rewardToOwner) (contracts/
e.sol#554)
BirdOracle.rewardProviders() (contracts/flatOracle.sol#538-561) ignores return value by birdToken.transfer(providers[i],rewardToEachProvider)
ts/flatOracle.sol#558)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```



```
BirdOracle.sendPayment() (contracts/flatOracle.sol#523-534) uses timestamp for comparisons
        Dangerous comparisons:
        - now < dueDate (contracts/flatOracle.sol#529)
BirdOracle.rewardProviders() (contracts/flatOracle.sol#538-561) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(timeAfterRewarded > 86400,You can call reward providers once in
BirdOracle.isApproved(address) (contracts/flatOracle.sol#563-565) uses timestamp for comparison
        Dangerous comparisons:
        - now < dueDateOf[_addr] (contracts/flatOracle.sol#564)
BirdOracle.isApproved() (contracts/flatOracle.sol#567-569) uses timestamp for comparisons
        Dangerous comparisons:
        - now < dueDateOf[msg.sender] (contracts/flatOracle.sol#568)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



```
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/flatOracle.sol#162-165)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/flatOracle.sol#171-175)
addProvider(address) should be declared external:
        - BirdOracle.addProvider(address) (contracts/flatOracle.sol#422-430)
removeProvider(address) should be declared external:
        - BirdOracle.removeProvider(address) (contracts/flatOracle.sol#432-439)
newChainRequest(address,string) should be declared external:
        - BirdOracle.newChainRequest(address,string) (contracts/flatOracle.sol#441-460)
updatedChainRequest(uint256,uint256) should be declared external:
        - BirdOracle.updatedChainRequest(uint256,uint256) (contracts/flatOracle.sol#463-489)
getRatingByAddress(address) should be declared external:
        - BirdOracle.getRatingByAddress(address) (contracts/flatOracle.sol#491-498)
getRating() should be declared external:
        - BirdOracle.getRating() (contracts/flatOracle.sol#500-502)
getProviders() should be declared external:
        - BirdOracle.getProviders() (contracts/flatOracle.sol#505-515)
sendPayment() should be declared external:
        - BirdOracle.sendPayment() (contracts/flatOracle.sol#523-534)
rewardProviders() should be declared external:
        - BirdOracle.rewardProviders() (contracts/flatOracle.sol#538-561)
isApproved(address) should be declared external:
        - BirdOracle.isApproved(address) (contracts/flatOracle.sol#563-565)
setMinConsensus(uint256) should be declared external:
        - BirdOracle.setMinConsensus(uint256) (contracts/flatOracle.sol#571-573)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

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.

Concluding Remarks

While conducting the audits of Bird Money 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 Bird Money developers. Resolving the areas of recommendations are up to the team'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 the Bird Money platform or its product neither 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 Pvt Ltd.