BiggerMINDS

Smart Contract Audit Report

September 26, 2022

Introduction

1. About BiggerMINDS

BiggerMINDS develops a unique set of complementary products, MIND+, NFT Marketplace, NFT Staking, NFTs, and more, to create an entire sustainable ecosystem, enabling users, businesses, and projects access to a token with unlimited utility in a vast ecosystem. In addition, by integrating MIND+ into all BiggerMINDS and none-BiggerMINDS products, the transactional volume increases, feeding the ecosystem and rewarding thousands of users from collected taxes.

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

2. About ImmuneBytes

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

The team has been able to secure 175+ 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 BiggerMINDS team has provided the following doc for the purpose of audit:

1. https://docs.google.com/document/d/1MgKKhNNppkzkGOg1Am0Q6bWS4YZJ-rSuRSPa78eQWGk/edit
2. https://blog.biggerminds.io/mind-a9a442268534
3. https://immunebytes.notion.site/Standard-Contract-BiggerMIND-97df1fa9ffed45d7923d977236d5860c

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 to find 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, including -

1. Structural analysis of the smart contract is checked and verified.
2. An extensive automated testing of all the contracts under scope is conducted.
3. Line-by-line Manual Code review is conducted to evaluate, analyze and identify the potential security risks in the contract.
4. Evaluation of the contract's intended behavior and the documentation shared is an imperative step to verify the contract behaves as expected.
5. For complex and heavy contracts, adequate integration testing is conducted to ensure that contracts perform acceptably while interacting.
6. Storage layout verifications in the upgradeable contract are a must.
7. An important step in the audit procedure is highlighting and recommending better gas optimization techniques in the contract.

Audit Details

- Project Name: BiggerMINDS
- Contracts Name: BRAINManagementUpgradeable, BRAINRewardsUpgradeable, BRAINNFTUpgradeable, BrainMaintainanceUpgradeable
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github link: https://github.com/BiggerMINDS/biggerminds_brains_v3
- Branch: main
- Commit hash: 72e606bc71b981a29fa2aaba6bad52191d4bf609
- Commit hash(final): 80ad8b1f0fc99134a31435d15928c6fb2222c1a1
- 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 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	-	2
Acknowledged	-	1	5

Finding Overview

S.no	Findings	Risk	Contract Name
1.	payMaintenanceFees() function has a broken logic. The refundable token amount is wrongly calculated & becomes more than it should be.	High	BRAINManagementUpgradeable
2.	Brain Users shall never be able to own the maximum approved amount of NFTs	High	BRAINManagementUpgradeable
3.	No Events emitted after imperative State Variable modification	Low	BRAINManagementUpgradeable
4.	Unused State variable found in contract	Low	BRAINManagementUpgradeable
5.	sendRewards() function violates all reward transfer checkpoints	Medium	BRAINRewardsUpgradeable
6.	Absence of Zero Address Validation before initiating withdrawal of tokens to treasury	Low	BRAINRewardsUpgradeable
7.	No Events emitted after imperative State Variable modification	Low	BRAINRewardsUpgradeable
8.	Redundant use of onlyApproved modifiers in functions	Informational	BRAINRewardsUpgradeable
9.	No upper threshold validation was found for the uint256 argument being used in loop	Low	BRAINNFTUpgradeable
10.	setNFTNumericInfo() and _setNFTStringInfo functions doesn't validate the propertyNum argument being passed	Low	BRAINNFTUpgradeable
11.	Coding Style Issues in the Contract	Informational	BRAINNFTUpgradeable
12.	No Events emitted after imperative State Variable modification in setInterval() function	Low	BrainMaintainanceUpgradeable

Contract Name: BRAINManagementUpgradeable

High Severity Issues

1. **payMaintenanceFees() function has a broken logic. The refundable token amount is wrongly calculated & becomes more than it should be.**
   Line no: 327-340

   Description:
   As per the current design of the payMaintenanceFees() function of the BRAINManagement contract, it provides an option for users to pay using their pending rewards by passing a boolean value of true for the usePendingRewards argument.

   If a user selects this option and at a particular instance their rewards are greater than the fee amount to be paid, the IF statement from lines 332-336 is executed. The intended behavior, as it seems, is for the user to be able to pay using their reward amount and receive a refund for the remaining token amount since their rewards are greater than the amount to be paid.

```
329         if (usePendingRewards) {
330             uint256 rewards = brainRewardsContract
331                 .payMaintenanceFeesWithRewards(brainIDs);
332             if (rewards > amount) {
333                 brainRewardsContract.sendRewards(address(this), amount);
334                 amount = 0;
335                 refund = rewards - amount; //@audit -> Wrong order of execution
336             } else {
```

   However, this IF statement block has a major bug due to the way it includes the order of execution.
   The function calculates the refundable amount (at Line 335) after assigning a zero value to the amount variable (at Line 334). This means the calculation of the refundable amount will always be equal to the amount of the total reward without taking into consideration the fee amount that was already paid from that specific reward amount. This is because the amount variable is assigned a zero value before calculating the total refundable amount for the user, and will therefore refund more tokens to the user than it actually should.

   Recommendation:
   There are 2 imperative recommendations for resolving this issue:
   a. Function redesign: The function should be redesigned to include the right order of execution so that the correct value of rewards, fee amount, as well as the refundable amount, is achieved before initiating any token transfers.

b. Including adequate test cases: It should be noted that in the current test scripts, not a single one of them covers the scenario where users are trying to pay maintenance fees using their pending rewards. This leads to an unwanted scenario of leaving out the edge cases while testing the behavior of a function.

As of now, every test case written for the payMaintenanceFees() function simply assumes the usePendingReward argument to always be false. Therefore, it fails to test the possible scenarios of the IF statement block from Line 329-340.

```
822         await ethers.provider.send("evm_mine");
823
824         await management.connect(user1).payMaintenanceFees([0], [1], false);
825         await management.connect(user2).payMaintenanceFees([1], [1], false);
826
```

It is highly recommended to include test cases that cover all possible edge case scenarios, especially when token transfers are involved.

Amended: The team has fixed the issue, and it is no longer present in the code.

2. **Brain Users shall never be able to own the maximum approved amount of NFTs**
   Line no - 541

Description:
The _mint() function includes an imperative criterion that states that a user shouldn't be able to mint more than the maximum approved brains allowed for minting.

This maximum brain amount is stored in the maxBRAINs state variable and can be updated by the approved addresses.
However, the IF statement in the mint function at the above-mentioned line number wrongly includes a condition that will never allow users to mint the maximum approved brain amount for minting.

```
ftrace | funcSig
535     function _mint( //@audit HIGH SEVERITY -> RECHECK
536         address recipient⬆,
537         string[] calldata names⬆,
538         uint256 amount⬆,
539         uint256 tokenAmount⬆
540     ) private {
541         if (brainNFTContract.getOwnerCount() + amount⬆ >= maxBRAINs) //@audit - Wrong condition
542             revert ExceedsMaxMint();
```

The condition in the contract states that the owner count of the user should never be greater than or equal to the maxBRAINs state variable, which means the user will not be able to mint the maximum allowed BRAIN NFTs but always 1 lesser than the actual max NFTs allowance.

Proof-Of-Concept (POC):
Kindly execute the test case provided below to validate the above-mentioned issue.
In the following test case:

- The maxBRAINs is set to 3. ()Thus allowing a user to mint and own a maximum of 3 BRAIN NFTs.
- However, when the user tries to mint 3 NFTs, the function fails with ExceedsMaxMint() error.

```
it("Create BRAINS with tokens", async function() {
        await management.setMaxBRAINs(3);
        await management.setBRAINPrice(brainPrice);
        expect(await management.brainPrice()).to.equal(100);
        await management.setPaymentToken(paymentToken.address);
        await expect(() =>
           management
              .connect(user1)
              .createBRAINsWithTokens(["AAAA", "BBBB", "CCCC"], 3)
        ).to.changeTokenBalances(
           paymentToken,
           [rewards, user1],
           [brainPrice * 3, -brainPrice * 3]
        );

console.log("--POC => Max Brains Amount Not Mintable---");
        const totalOwners = await brainNFT.getOwnerCount();
        console.log("Total Owner Count for User-1",totalOwners.toString())

        expect(await management.totalSupply()).to.equal(3);
        expect(await management.balanceOf(user1.address)).to.equal(3);
        let names = await management.getBRAINNames([0, 1, 2]);
        expect(names.length).to.equal(3);
        expect(names[0]).to.equal("AAAA");
        expect(names[1]).to.equal("BBBB");
      });
```

Amended: The team has fixed the issue, and it is no longer present in the code.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **No Events emitted after imperative State Variable modification**
   Line no: 473-500

   Description:
   Functions that update imperative arithmetic state variables of the contract should emit an event after its execution.

   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.

   Acknowledged: The team has acknowledged the issue.

2. **Unused State variable found in contract**
   Line no - 34

   Description:
   The contract includes a state variable, i.e., synapseContract but never uses it throughout the contract.

   Recommendation:
   Unwanted state variables, functions or modifiers, etc, must be removed from the contract to optimize gas and enhance readability.

   Amended: The team has fixed the issue, and it is no longer present in the code.

Recommendation / Informational

No issues were found.

Contract Name: BRAINRewardsUpgradeable

High Severity Issues

No issues were found.

Medium Severity Issues

1. **sendRewards() function violates all reward transfer checkpoints**
   Line no: 155-160

   Description:
   As per the current architecture of the BRAINRewards contract, the transfer of reward tokens to users should only happen if some specific criteria are met.

   For instance, the rewards should only be claimable if:
   - The user is actually an owner of the brainNFT, or
   - The maintenance fee must not be paid past the grace period, etc.

   However, the contract also includes an additional function, i.e., sendRewards() which allows any approved address to send reward tokens to any given address. It doesn't include any checkpoints to ensure whether or not the account actually holds a brainNFT in the first place.

```
ftrace | funcSig
155  function sendRewards(address account⬆, uint256 rewards⬆) //@audit -> No validations before sending reward tokens
156      external
157      onlyApproved
158  {   ! EXTCALL !
159      IERC20Upgradeable(rewardToken).safeTransfer(account⬆, rewards⬆);
160  }
161
```

   Although the function has been assigned an access control modifier, it doesn't adhere to the reward token transfer rules of the contract.

   Recommendation:
   A function that deals with token or payment transfer must follow the contract's intended behavior and include necessary checkpoints even if it's an onlyOwner/onlyApproved function.

   Acknowledged: The team has acknowledged the issue.

Low Severity Issues

1. **Absence of Zero Address Validation before initiating withdrawal of tokens to treasury**

   Description:
   The contract includes functions like withdrawNativeToTreasury or withdrawTokensToTreasury, which play a significant role in carrying out withdrawals of tokens or ether from the contract.

   However, during the review, it was found that the functions do not validate whether or not the brainManagementContract address, which is passed as the treasury address, is adequately updated with the right address before this function is triggered.

   Recommendation:
   A require statement should be included in such functions to ensure no zero address is passed in the arguments.

   Acknowledged: The team has acknowledged the issue.

2. **No Events emitted after imperative State Variable modification**
   Line no: 211, 216, 219
   Description:

   Functions that update imperative arithmetic state variables of the contract should emit an event after its execution.

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

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

   Acknowledged: The team has acknowledged the issue.

Recommendation / Informational

1. **Redundant use of onlyApproved modifiers in functions**
   Line no - 82, 108, 164, 173, 182, 191, 199

Description:
As per the current design of the contracts, the _update() private function is used as a modifier in several other functions. This private function also includes the onlyApproved modifier.

However, all the functions in the above-mentioned line numbers redundantly include the onlyApproved modifier as well despite it being already attached with the _update() function.

This increases the gas consumption of the functions during execution.

Recommendation:
Avoid the use of redundant functions or modifiers in the contract.

Amended: The team has fixed the issue, and it is no longer present in the code.

Contract Name: BRAINNFTUpgradeable

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **No upper threshold validation was found for the uint256 argument being used in loop**
   Line no: 57-73

   Description:
   The batchMint() function allows the caller to pass any value for the amount of NFTs to be minted without any input validation on the upper threshold for the same.

```
62        uint256[] memory nftIDs = new uint256[](amount);
63        for (uint256 x; x < amount; x++) { // @audit
64            string memory name = _names[x];
65            _validateName(name);
66            _mint(receiver);
67            uint256 id = _owners.length - 1;
68            nftIDs[x] = id;
69            brainsToNames[id] = name;
70            names[name] = true;
71        }
```

   It's imperative to note that the for loop of this function relies on the amount argument as the number of iterations depends on it. Therefore, it makes it quite significant to validate the arguments being passed as the amount even if the function is assigned an onlyProxy modifier.

   Recommendation:
   Adequate input validations must be included for important arguments being passed to the function.

   Acknowledged: The team has acknowledged the issue.

2. **setNFTNumericInfo() and _setNFTStringInfo functions doesn't validate the propertyNum argument being passed**
   Line no: 43-56

   Description:
   The NFTInfo struct in the NFTInfoUpgradeable contract only includes 3 propertyNum members.

   However, the setNFTNumbercInfo() as well as the _setNFTStringInfo() function in the BRAINNFTUpgradeable contract doesn't include any proper input validations for the propertyNum argument being passed to the function body.

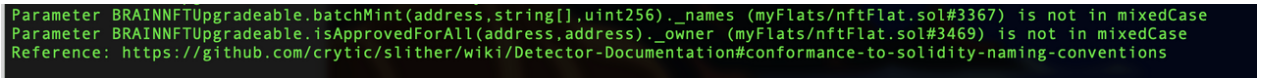
   As per the current function body, any value can be passed as the propertyNum argument, which is simply aligned with the else block of the function. This doesn't represent an adequate design.

   Recommendation:
   Include input validations in the function to ensure only the right arguments are passed.

   Amended: The team has fixed the issue, and it is no longer present in the code.

Recommendation / Informational

1. **Coding Style Issues in the Contract**
   Explanation:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and, in some specific scenarios, may lead to bugs in the future.



   During the automated testing, it was found that the BRAINNFTUpgradeable contract had quite a few code-style issues.

   Recommendation:
   Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

   Amended: The team has fixed the issue, and it is no longer present in the code.

Contract Name: BrainMaintainanceUpgradeable

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **No Events emitted after imperative State Variable modification in setInterval() function**
   Line no: 148

   Description:
   Functions that update imperative arithmetic state variables of the contract should emit an event after its execution.

   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.

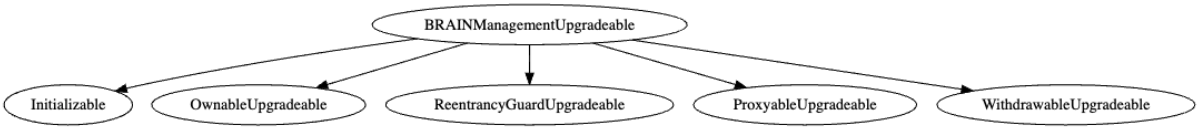
   Acknowledged: The team has acknowledged the issue.

Recommendation / Informational

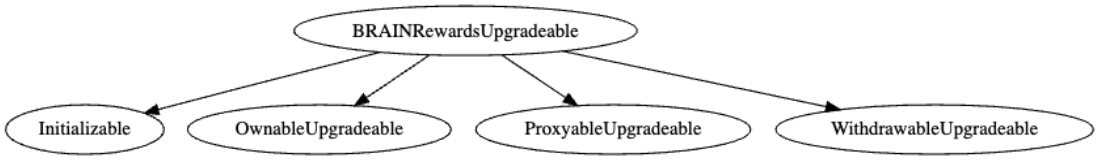
No issues were found.

Additional Details - Inheritance Graph

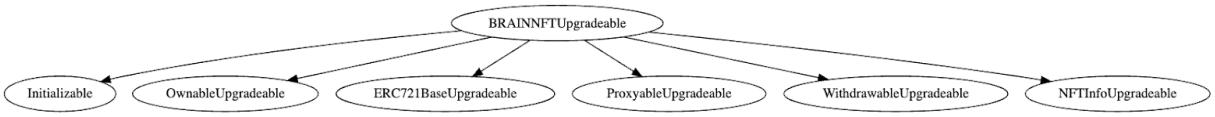
1. BRAINManagementUpgradeable



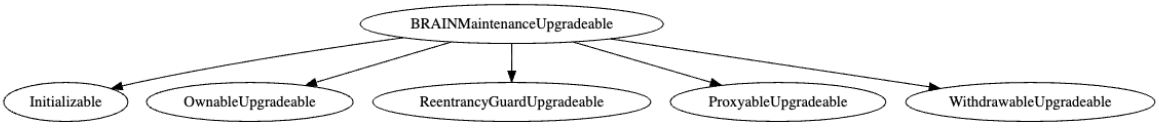
2. BRAINRewardsUpgradeable



3. BRAINNFTUpgradeable

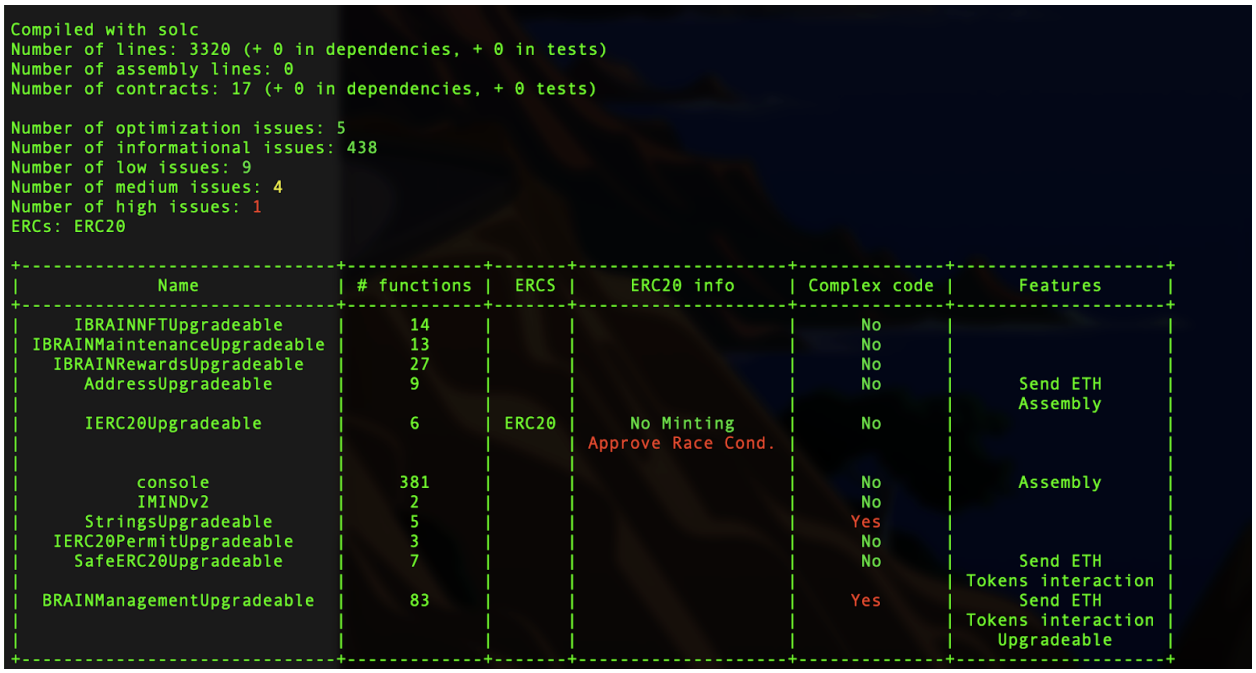


4. BRAINMaintenanceUpgradeable



Automated Audit Result

1. BRAINManagementUpgradeable

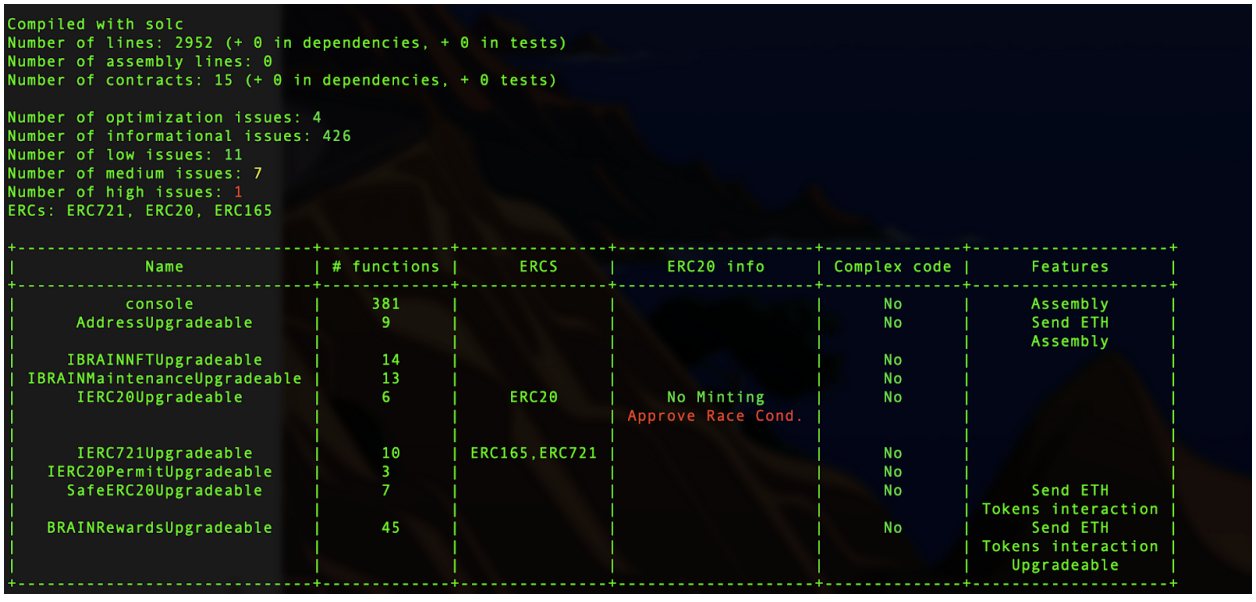


```
Compiled with solc
Number of lines: 3320 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 17 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 438
Number of low issues: 9
Number of medium issues: 4
Number of high issues: 1
ERCs: ERC20

+-------------------------------+-------------+-------+--------------------+--------------+--------------------+
|             Name              | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+-------------------------------+-------------+-------+--------------------+--------------+--------------------+
|     IBRAINNFTUpgradeable      |      14     |       |                    |      No      |                    |
|  IBRAINMaintenanceUpgradeable |      13     |       |                    |      No      |                    |
|    IBRAINRewardsUpgradeable   |      27     |       |                    |      No      |                    |
|       AddressUpgradeable      |      9      |       |                    |      No      |      Send ETH      |
|                               |             |       |                    |              |      Assembly      |
|       IERC20Upgradeable       |      6      | ERC20 |     No Minting     |      No      |                    |
|                               |             |       | Approve Race Cond. |              |                    |
|                               |             |       |                    |              |                    |
|            console            |     381     |       |                    |      No      |      Assembly      |
|            IMINDv2            |      2      |       |                    |      No      |                    |
|       StringsUpgradeable      |      5      |       |                    |     Yes      |                    |
|    IERC20PermitUpgradeable    |      3      |       |                    |      No      |                    |
|      SafeERC20Upgradeable     |      7      |       |                    |      No      |      Send ETH      |
|                               |             |       |                    |              | Tokens interaction |
|   BRAINManagementUpgradeable  |      83     |       |                    |     Yes      |      Send ETH      |
|                               |             |       |                    |              | Tokens interaction |
|                               |             |       |                    |              |    Upgradeable     |
+-------------------------------+-------------+-------+--------------------+--------------+--------------------+
```

2. BRAINRewardsUpgradeable



```
Compiled with solc
Number of lines: 2952 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 15 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 4
Number of informational issues: 426
Number of low issues: 11
Number of medium issues: 7
Number of high issues: 1
ERCs: ERC721, ERC20, ERC165

+-------------------------------+-------------+---------------+--------------------+--------------+--------------------+
|             Name              | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+-------------------------------+-------------+---------------+--------------------+--------------+--------------------+
|            console            |     381     |               |                    |      No      |      Assembly      |
|       AddressUpgradeable      |      9      |               |                    |      No      |      Send ETH      |
|                               |             |               |                    |              |      Assembly      |
|     IBRAINNFTUpgradeable      |      14     |               |                    |      No      |                    |
|  IBRAINMaintenanceUpgradeable |      13     |               |                    |      No      |                    |
|       IERC20Upgradeable       |      6      |     ERC20     |     No Minting     |      No      |                    |
|                               |             |               | Approve Race Cond. |              |                    |
|                               |             |               |                    |              |                    |
|       IERC721Upgradeable      |      10     | ERC165,ERC721 |                    |      No      |                    |
|    IERC20PermitUpgradeable    |      3      |               |                    |      No      |                    |
|      SafeERC20Upgradeable     |      7      |               |                    |      No      |      Send ETH      |
|                               |             |               |                    |              | Tokens interaction |
|    BRAINRewardsUpgradeable    |      45     |               |                    |      No      |      Send ETH      |
|                               |             |               |                    |              | Tokens interaction |
|                               |             |               |                    |              |    Upgradeable     |
+-------------------------------+-------------+---------------+--------------------+--------------+--------------------+
```

3. BRAINNFTUpgradeable

```
Compiled with solc
Number of lines: 3501 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 18 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 14
Number of informational issues: 467
Number of low issues: 17
Number of medium issues: 12
Number of high issues: 1
ERCs: ERC721, ERC20, ERC165

+-----------------------------+-------------+---------------+--------------------+--------------+--------------------+
|            Name             | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+-----------------------------+-------------+---------------+--------------------+--------------+--------------------+
|     AddressUpgradeable      |      9      |               |                    |      No      |      Send ETH      |
|                             |             |               |                    |              |      Assembly      |
|     IERC20Upgradeable       |      6      |     ERC20     |     No Minting     |      No      |                    |
|                             |             |               | Approve Race Cond. |              |                    |
|                             |             |               |                    |              |                    |
|          console            |     381     |               |                    |      No      |      Assembly      |
|     StringsUpgradeable      |      5      |               |                    |     Yes      |                    |
| IERC721ReceiverUpgradeable  |      1      |               |                    |      No      |                    |
|    BRAINNFTUpgradeable      |     101     | ERC165,ERC721 |                    |      No      |      Send ETH      |
|                             |             |               |                    |              | Tokens interaction |
|                             |             |               |                    |              |      Assembly      |
|                             |             |               |                    |              |    Upgradeable     |
+-----------------------------+-------------+---------------+--------------------+--------------+--------------------+
```

4. BRAINMaintenanceUpgradeable

```
Compiled with solc
Number of lines: 3898 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 23 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 15
Number of informational issues: 475
Number of low issues: 13
Number of medium issues: 15
Number of high issues: 1
ERCs: ERC20, ERC165, ERC721

+-----------------------------+-------------+---------------+--------------------+--------------+--------------------+
|            Name             | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+-----------------------------+-------------+---------------+--------------------+--------------+--------------------+
|      AddressUpgradeable     |      9      |               |                    |      No      |      Send ETH      |
|                             |             |               |                    |              |      Assembly      |
|      StringsUpgradeable     |      5      |               |                    |     Yes      |                    |
|  IERC721ReceiverUpgradeable |      1      |               |                    |      No      |                    |
|    ERC721BaseUpgradeable    |      72     | ERC165,ERC721 |                    |      No      |      Assembly      |
|                             |             |               |                    |              |    Upgradeable     |
|          console            |     381     |               |                    |      No      |      Assembly      |
|      IERC20Upgradeable      |      6      |     ERC20     |     No Minting     |      No      |                    |
|                             |             |               | Approve Race Cond. |              |                    |
|                             |             |               |                    |              |                    |
|      NFTInfoUpgradeable     |      8      |               |                    |      No      |    Upgradeable     |
|     IBRAINNFTUpgradeable    |      14     |               |                    |      No      |                    |
|   IBRAINRewardsUpgradeable  |      27     |               |                    |      No      |                    |
|   IERC20PermitUpgradeable   |      3      |               |                    |      No      |                    |
|     SafeERC20Upgradeable    |      7      |               |                    |      No      |      Send ETH      |
|                             |             |               |                    |              | Tokens interaction |
| BRAINMaintenanceUpgradeable |      34     |               |                    |      No      |      Send ETH      |
|                             |             |               |                    |              | Tokens interaction |
|                             |             |               |                    |              |    Upgradeable     |
+-----------------------------+-------------+---------------+--------------------+--------------+--------------------+
myFlats/maintainanceFlat.sol analyzed (23 contracts)
```

Auditor Test Cases

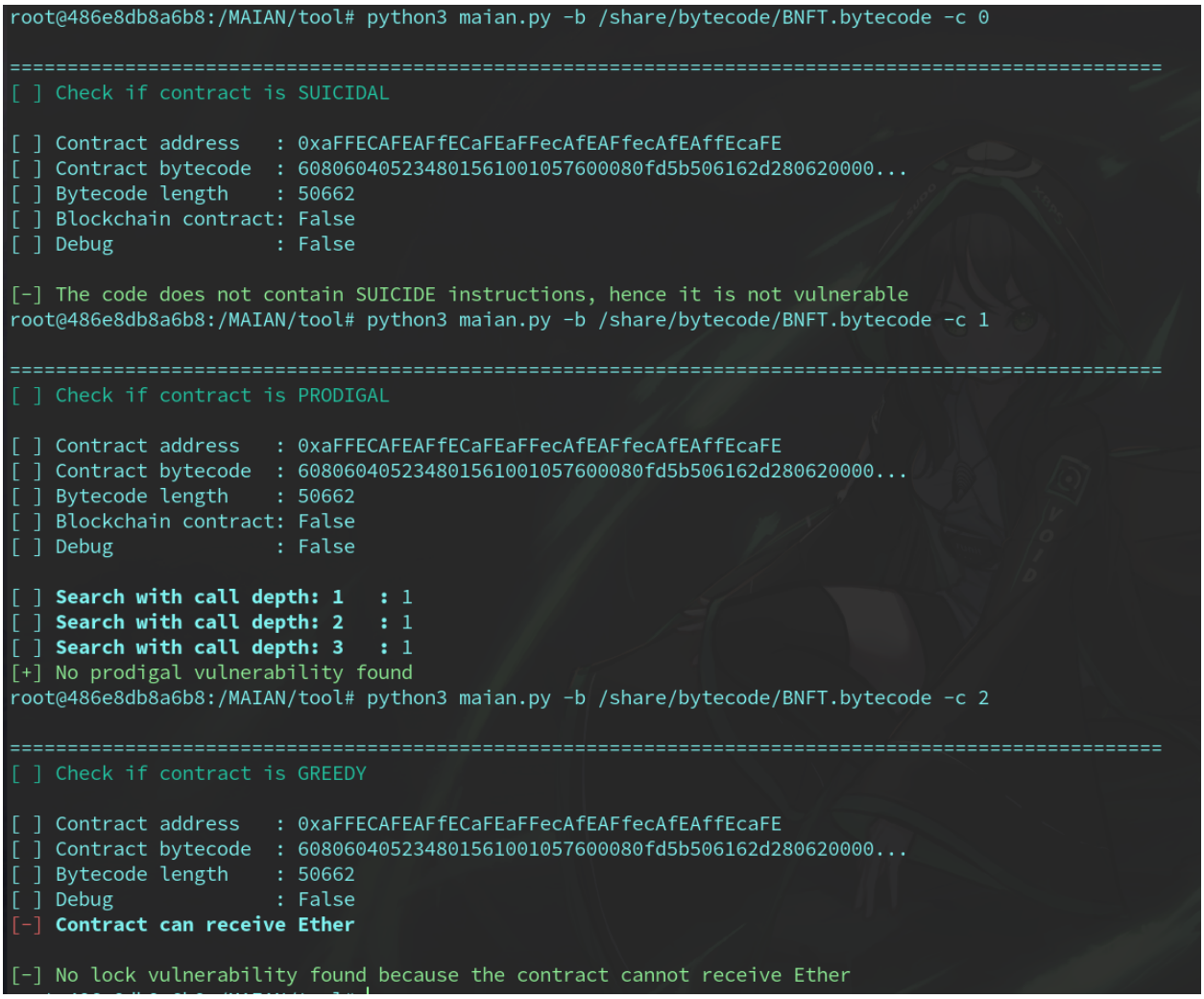
```
Running 5 tests for test/BRAINNFTUpgradeable.t.sol:BRAINNFTUpgradeableTest
[PASS] test_LessThen_isValidName() (gas: 9010)
[PASS] test_batchMint() (gas: 179302)
[PASS] test_getBRAINNames() (gas: 28734)
[PASS] test_isValidName() (gas: 9678)
[PASS] test_renameBRAIN() (gas: 191608)
Test result: ok. 5 passed; 0 failed; finished in 6.12ms
```

```
Running 4 tests for test/BRAINManagementUpgradeable.t.sol:BRAINManagementUpgradeableTest
[PASS] test_initializeReferences() (gas: 434981)
[PASS] test_mint() (gas: 1862378)
[PASS] test_payMaintenanceFees_false() (gas: 1037585)
[PASS] test_payMaintenanceFees_true() (gas: 1076165)
Test result: ok. 4 passed; 0 failed; finished in 9.08ms
```

Fuzz Testing

```
Running 4 tests for test/BRAINManagementUpgradeable.t.sol:BRAINManagementUpgradeableTest
[PASS] test_initializeReferences() (gas: 435081)
[PASS] test_mint(string[],uint256) (runs: 256, μ: 1915054, ~: 1914440)
[PASS] test_payMaintenanceFees_false(uint256[],uint256[]) (runs: 256, μ: 1050896, ~: 1050651)
[PASS] test_payMaintenanceFees_true(uint256[],uint256[]) (runs: 256, μ: 1095600, ~: 1095603)
Test result: ok. 4 passed; 0 failed; finished in 390.45ms
```

Maian



```
root@486e8db8a6b8:/MAIAN/tool# python3 maian.py -b /share/bytecode/BReward.bytecode -c 0

================================================================================================================
[ ] Check if contract is SUICIDAL

[ ] Contract address   : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode  : 6080604052348015610010057600080fd5b50613bb980610020...
[ ] Bytecode length    : 30642
[ ] Blockchain contract: False
[ ] Debug              : False

[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@486e8db8a6b8:/MAIAN/tool# python3 maian.py -b /share/bytecode/BReward.bytecode -c 1

================================================================================================================
[ ] Check if contract is PRODIGAL

[ ] Contract address   : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode  : 6080604052348015610010057600080fd5b50613bb980610020...
[ ] Bytecode length    : 30642
[ ] Blockchain contract: False
[ ] Debug              : False

[ ] Search with call depth: 1   : 1
[ ] Search with call depth: 2   : 1
[ ] Search with call depth: 3   : 1
[+] No prodigal vulnerability found
root@486e8db8a6b8:/MAIAN/tool# python3 maian.py -b /share/bytecode/BReward.bytecode -c 2

================================================================================================================
[ ] Check if contract is GREEDY

[ ] Contract address   : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode  : 6080604052348015610010057600080fd5b50613bb980610020...
[ ] Bytecode length    : 30642
[ ] Debug              : False
[-] Contract can receive Ether

[-] No lock vulnerability found because the contract cannot receive Ether
```

```
root@486e8db8a6b8:/MAIAN/tool# python3 maian.py -b /share/bytecode/BManagement.bytecode -c 0

======================================================================================================
[ ] Check if contract is SUICIDAL

[ ] Contract address    : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode   : 6080604052348015620000115760008 0fd5b50618fdc806200...
[ ] Bytecode length     : 73724
[ ] Blockchain contract: False
[ ] Debug               : False

[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@486e8db8a6b8:/MAIAN/tool# python3 maian.py -b /share/bytecode/BManagement.bytecode -c 1

======================================================================================================
[ ] Check if contract is PRODIGAL

[ ] Contract address    : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode   : 6080604052348015620000115760008 0fd5b50618fdc806200...
[ ] Bytecode length     : 73724
[ ] Blockchain contract: False
[ ] Debug               : False

[ ] Search with call depth: 1   : 1
[ ] Search with call depth: 2   : 1
[ ] Search with call depth: 3   : 1
[+] No prodigal vulnerability found
root@486e8db8a6b8:/MAIAN/tool# python3 maian.py -b /share/bytecode/BManagement.bytecode -c 2

======================================================================================================
[ ] Check if contract is GREEDY

[ ] Contract address    : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode   : 6080604052348015620000115760008 0fd5b50618fdc806200...
[ ] Bytecode length     : 73724
[ ] Debug               : False
[-] Contract can receive Ether

[-] No lock vulnerability found because the contract cannot receive Ether
```

Concluding Remarks

While conducting the audits of the BiggerMINDS 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 BiggerMINDS 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