

**Lab Astral**

**MYRT Token**

**Smart Contract Audit  
Final Report**



**May 18, 2022**

<b>Introduction</b>	<b>3</b>
About Lab Astral	3
About ImmuneBytes	3
<b>Documentation Details</b>	<b>3</b>
<b>Audit Process &amp; Methodology</b>	<b>4</b>
<b>Audit Details</b>	<b>4</b>
<b>Audit Goals</b>	<b>5</b>
<b>Security Level Reference</b>	<b>5</b>
High Severity Issues	6
Medium Severity Issues	6
Low Severity Issues	6
<b>Recommendation / Informational</b>	<b>7</b>
<b>Unit Tests</b>	<b>8</b>
<b>Automated Audit Result</b>	<b>9</b>
Slither	9
Maian Bytecode Analysis	9
Maian Solidity Code Analysis	10
Mythril	10
Echidna	11
Echidna Gas Test	12
<b>Concluding Remarks</b>	<b>13</b>
<b>Disclaimer</b>	<b>13</b>

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 Lab Astral

Lab Astral is a world-class team of experts in computer science, economics, and finance, Based in Malaysia, Lab Astral provides cutting-edge blockchain research, and technical and project solutions with key capabilities in infrastructure, Sinartootreots, and marketplace automation.

Currently, Lab Astral is developing new blockchain consensus models that are more energy efficient including Proof of Stake and Proof of History, etc, layer 2 protocols for private blockchains to integrate with popular main nets in the market, and smart contracts with or without /PFC integrations. These developments will provide the framework, and protocols for wob3 to be plugged and play with existing web2 assets.

Visit <https://labastral.com/> to know more about it.

### 2. About ImmuneBytes

ImmuneBytes is a security start-up to provides 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, and dydx.

The team has been able to secure 105+ 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 Lab Astral team has provided the following doc for the purpose of audit:

1. Lab Astral MYRT Whitepaper (LA Replied 10.05.2022).docx.pdf
2. <https://labastral.com/documents/MYRT-Deck.pdf>

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: Lab Astral
- Contracts Name: myrt.sol
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github commits for audit: 461980f6d1ecfd43ac9c95e97b167caf824f8a2a
- 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	<u>High</u>	<u>Medium</u>	<u>Low</u>
Open	-	-	-
Closed	-	-	1

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

No issues were found.

## Medium Severity Issues

No issues were found.

## Low Severity Issues

### 1. External Visibility should be preferred

#### Explanation:

Those functions which are not called internally by the contract should be marked with external visibility instead of public visibility. This will effectively result in gas optimization.

#### Recommendation:

The mint and burn function can be refactored to external functions as follows:

```

1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.4.22 <0.9.0;
3  import "@openzeppelin/contracts/access/Ownable.sol";
4  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6  contract MYRT is ERC20, Ownable{
7      constructor () ERC20("MYRT", "myrt") Ownable(){}
8      |_mint(msg.sender, 25000000000 * ( 10 ** uint256(decimals())));
9  }
10     function mint(address account, uint256 amount) external onlyOwner {
11         |_mint(account, amount);
12     }
13     function burn(address account, uint256 amount) external onlyOwner {
14         |_burn(account, amount);
15     }
16 }
```

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.

## Recommendation / 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. Infinite Token Supply

Current implementation allows the owner to mint an unlimited amount of tokens. If this is not intentional, it is recommended to include a maximum supply for the token within the contract.

## Unit Tests

```
Compiling your contracts...
=====
> Compiling ./contracts/myrt.sol
> Compiling @openzeppelin/contracts/access/Ownable.sol
> Compiling @openzeppelin/contracts/token/ERC20/ERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/IERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
> Compiling @openzeppelin/contracts/utils/Context.sol
> Artifacts written to /var/folders/jz/tqv1m4pd7zzdhyz78f_685xm0000gn/T/test--41212-o2kLgEeImndk
> Compiled successfully using:
- solc: 0.8.10+commit.fc410830.Emscripten.clang

Contract: MYRT
✓ sender should has 25 billion MYRT
✓ owner address should be sender address
✓ owner should be able to burn MYRT (1043ms)
✓ other user should be unable to burn MYRT (1117ms)
✓ owner should be able to mint MYRT (1033ms)
✓ other user should be unable to mint MYRT (1028ms)
✓ should be able to transfer ownership (1037ms)
✓ other should not be able to transfer ownership (1048ms)
✓ owner should be null afer renounce ownership (1038ms)
✓ other should not be able to renounce ownership (1031ms)

10 passing (19s)
```

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

## Slither

```
Compiled with solc
Number of lines: 621 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 12
Number of informational issues: 10
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+-----+ +-----+ +-----+ +-----+
| Name | # functions | ERCS | ERC20 info | Complex code | Features |
+-----+ +-----+ +-----+ +-----+
| MYRT | 38 | ERC20 | ` Minting | No |
| | | | Approve Race Cond. | | |
+-----+ +-----+ +-----+ +-----+
```

## Maian Bytecode Analysis

```
root@83d2372a461d:/MAIAN/tool# python3 maian.py -b /share/flat.bytecode -c 0
=====
[ ] Check if contract is SUICIDAL
[ ] Contract address : 0xaFFECafeAFFEcAFeAFFfecAfEAFfEcaFE
[ ] Contract bytecode : 60806040523480156200001157600080fd5b50604051806040...
[ ] Bytecode length : 19938
[ ] Blockchain contract: False
[ ] Debug : False
[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@83d2372a461d:/MAIAN/tool# python3 maian.py -b /share/flat.bytecode -c 1
=====
[ ] Check if contract is PRODIGAL
[ ] Contract address : 0xaFFECafeAFFEcAFeAFFfecAfEAFfEcaFE
[ ] Contract bytecode : 60806040523480156200001157600080fd5b50604051806040...
[ ] Bytecode length : 19938
[ ] Blockchain contract: False
[ ] Debug : False
[+] The code does not have CALL/SUICIDE, hence it is not prodigal
root@83d2372a461d:/MAIAN/tool# python3 maian.py -b /share/flat.bytecode -c 2
=====
[ ] Check if contract is GREEDY
[ ] Contract address : 0xaFFECafeAFFEcAFeAFFfecAfEAFfEcaFE
[ ] Contract bytecode : 60806040523480156200001157600080fd5b50604051806040...
[ ] Bytecode length : 19938
[ ] Debug : False
[-] Contract can receive Ether
[-] No lock vulnerability found because the contract cannot receive Ether
```

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.

## Maian Solidity Code Analysis

```
root@83d2372a461d:/MAIAN/tool# python3 maian.py -s /share/flat.sol MYRT -c 0
=====
[ ] Compiling Solidity contract from the file /share/flat.sol ... Done
[ ] Connecting to PRIVATE blockchain emptychain . ESTABLISHED
[ ] Deploying contract confirmed at address: 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract code length on the blockchain : 7624 : 0x6080604052348015610010576000...
[ ] Contract address saved in file: ./out/MYRT.address
[ ] Check if contract is SUICIDAL

[ ] Contract address : 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract bytecode : 608060405234801561001057600080fd5b5060043610610100...
[ ] Bytecode length : 15248
[ ] Blockchain contract: True
[ ] Debug : False

[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@83d2372a461d:/MAIAN/tool# python3 maian.py -s /share/flat.sol MYRT -c 1
=====

[ ] Compiling Solidity contract from the file /share/flat.sol ... Done
[ ] Connecting to PRIVATE blockchain emptychain . ESTABLISHED
[ ] Sending Ether to contract 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306 ... tx[0] mined Sent!
[ ] Deploying contract confirmed at address: 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract code length on the blockchain : 7624 : 0x6080604052348015610010576000...
[ ] Contract address saved in file: ./out/MYRT.address
[ ] The contract balance: 44 Positive balance
[ ] Check if contract is PRODIGAL

[ ] Contract address : 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract bytecode : 608060405234801561001057600080fd5b5060043610610100...
[ ] Bytecode length : 15248
[ ] Blockchain contract: True
[ ] Debug : False
[+] The code does not have CALL/SUICIDE, hence it is not prodigal
root@83d2372a461d:/MAIAN/tool# python3 maian.py -s /share/flat.sol MYRT -c 2
=====

[ ] Compiling Solidity contract from the file /share/flat.sol ... Done
```

```
root@83d2372a461d:/MAIAN/tool# python3 maian.py -s /share/flat.sol MYRT -c 2
=====
[ ] Compiling Solidity contract from the file /share/flat.sol ... Done
[ ] Connecting to PRIVATE blockchain emptychain . ESTABLISHED
[ ] Deploying contract confirmed at address: 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract code length on the blockchain : 7624 : 0x6080604052348015610010576000...
[ ] Contract address saved in file: ./out/MYRT.address
[ ] Check if contract is GREEDY

[ ] Contract address : 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract bytecode : 608060405234801561001057600080fd5b5060043610610100...
[ ] Bytecode length : 15248
[ ] Debug : False
[-] Contract can receive Ether

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

## Mythril

The analysis was completed successfully. No issues were detected.

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.

## Echidna

```
Tests found: 2
Seed: -1292061514428537991
Unique instructions: 2203
Unique codehashes: 1
Corpus size: 4
Echidna 2.0.0
Tests
echidna_mint: fuzzing (1591/50000)
echidna_burn: fuzzing (1591/50000)

monityl/Fuzzing/Echidna
Latest commit of monityl on Jun 10, 2020
History
```

```
Tests found: 2
Seed: -1292061514428537991
Unique instructions: 3293
Unique codehashes: 1
Corpus size: 19
Echidna 2.0.0
Tests
echidna_mint: PASSED!
echidna_burn: PASSED!

Campaign complete, C-c or esc to exit
monityl
```

```
Analyzing contract: /home/danish/audit/MYRT/flat.sol:TestToken
echidna_burn: passed! []
echidna_mint: passed! []

Unique instructions: 3293
Unique codehashes: 1
Corpus size: 19
Seed: -1292061514428537991
Transactions | Deployments | Security | Status
```

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.

## Echidna Gas Test

```

Echidna 2.0.0
Tests found: 1
Seed: -3529227244288384689
Unique instructions: 2435
Unique codehashes: 1
Corpus size: 20
Tests-
echidna_gas_test: fuzzing (3164/50000)

echidna_gas_test: passed! []
Decimals used a maximum of 351 gas
Call sequence:
    decimals() Time delay: 98309 seconds Block delay: 39496
totalSupply used a maximum of 490 gas
Call sequence:
    totalSupply() Time delay: 526756 seconds Block delay: 31046
Owner used a maximum of 574 gas
Call sequence:
    owner() Time delay: 132550 seconds Block delay: 43095
Name used a maximum of 1451 gas
Call sequence:
    name() Time delay: 506628 seconds Block delay: 16859
Symbol used a maximum of 1516 gas
Call sequence:
    symbol() Time delay: 93469 seconds Block delay: 59982
renounceOwnership used a maximum of 2388 gas
Call sequence:
    renounceOwnership() Time delay: 488170 seconds Block delay: 15852
transferOwnership used a maximum of 2821 gas
Call sequence:
    transferOwnership(0x80a3ce15f87c4c3304d5ac9e61c2e5e7c589de3b) Time delay: 347538 seconds Block delay: 9
balanceOf used a maximum of 2915 gas
Call sequence:
    balanceOf(0x50baaacdac19236cb582529fc51ca3b3a984b101) Time delay: 2 seconds Block delay: 59246
allowance used a maximum of 3226 gas
Call sequence:
    allowance(0xd33018757b0cf079e315d7007c399be75033108,0x0) Time delay: 583144 seconds Block delay: 2925
allowance used a maximum of 3226 gas
Call sequence:
    allowance(0xd33018757b0ccf079e315d7007c399be75033108,0x0) Time delay: 583144 seconds Block delay: 2925
decreaseAllowance used a maximum of 5587 gas
Call sequence:
    decreaseAllowance(0x4fffa957466cad52b88da3556c1c399ff63b13d,0) Time delay: 349646 seconds Block delay: 5724
burn used a maximum of 5835 gas
Call sequence:
    burn(0x8e3cd3cdc93b4cc7f838bed062b31001de454308,0) Time delay: 592492 seconds Block delay: 17
transferFrom used a maximum of 13052 gas
Call sequence:
    transferFrom(0xd382a99f4bc21d5cf98f076e09a62a1892d59eb,0x20000,0) Time delay: 82671 seconds Block delay: 49770
approve used a maximum of 25264 gas
Call sequence:
    approve(0xf5f3096496e985550b88750d0255aafb0d6037e,4370000) Time delay: 494134 seconds Block delay: 44650
increaseAllowance used a maximum of 25637 gas
Call sequence:
    increaseAllowance(0xd3a14e4704e30e2b29e89730bf0e579d4282d567,115792089237316195423570985008687907853269984665640564039457584007888129639936) Time delay: 513837 seconds Block delay: 15728
mint used a maximum of 25858 gas
Call sequence:
    mint(0x3282e9170643a7bca33f27aaed86b43322535d,4370001) Time delay: 512679 seconds Block delay: 59443
transfer used a maximum of 25918 gas
Call sequence:
    transfer(0x3faa46807e49bbead18df290a629b6969a6140db,4370001) Time delay: 136069 seconds Block delay: 49682
Unique instructions: 3293
Unique codehashes: 1
Corpus size: 33
Seed: -1662330538730848948

```

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 the Lab Astral smart contract, it was observed that the contracts contain only one Low severity issue.

Our auditors suggest that the Low severity issue 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 Lab Astral 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*

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.