OnGrid Systems

SMART CONTRACT AUDIT REPORT for METARUN BSC BEP-20 TOKEN

Yerevan, AM
April 12, 2022

Document Properties

Client	Metarun
Title	Smart contract Audit Report
Document Version	v1.0.0
Consultants engaged	2
Person-hours spent	30
Request date	April 3, 2022
Revision date	April 12, 2022
Classification	Public

Contact

For more information about this document and its contents, please contact On-Grid security team:

Site	https://ongrid.pro
Email	info@ongrid.pro
Phone	+374 (99) 46-46-47

Table of Contents

General provisions

In March 2022, Metarun GameFi project engaged OnGrid team to perform a code review of the ERC-20 token smart contract. Given the opportunity to review the Metarun token smart contract source code, we outline in the report our approach to identify potential security issues in the smart contract implementation, expose inconsistencies between its code and specification, and provide additional suggestions or recommendations for its improvement.

About MetaRun ERC-20 token

MetaRun BEP-20 token is ERC-20 compatible core fungible asset of Metarun GameFi Play-to-Earn project. At the time of the check, the token contract was already deployed on BNB chain (ex Binance Smart Chain), partially minted, distributed or vested as the project's main asset. The basic information of MetaRun BEP-20 token is as follows:

Platform	EVM
Language	Solidity
Token Standard	BEP-20 (ERC-20 compatible)
Name	METARUN
Symbol	MRUN
Supply	Mintable, Capped at 1,000,000,000 Current Supply: 713,213,000
Contract specification	See README.md in the repository root
Compiler version and options	v0.8.11+commit.d7f03943 No optimization
Deployed on (Network, address)	BNB chain (ex Binance Smart Chain) 0xCa0D640a401406f3405b4C252a5d0c4d17F38EBb Code verified on BscScan

Audit Scope

In this section we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report. The assessment was performed on the source code files listed in the table below:

Repository Name	metarun-contracts
Repository URL	https://github.com/metarungame/metarun-contracts
Branch	master
Commit Hash and date	6b777c75cceb1b27219555eceffe3b1e796e8187 April 6, 2022
Specification	README.md
Contract Files	contracts/MetarunToken.sol

Imported libraries	@openzeppelin/contracts/token/ERC20/ERC20.sol @openzeppelin/contracts/access/AccessControl.sol @openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol @openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol
Relevant tests	MetarunToken.test.js
Audit Method	WhiteBox, Manual + Automated

All files and artefacts not explicitly listed above are out of scope.

Assessment Methodology

Our team performed manual analysis of code functionality using Hardhat test framework (with tests provided by implementers) and its debugger, manual code inspection using VSCode and Remix IDEs and Surya, utility tool for smart contract systems which provides a number of useful visual outputs and information about the structure of smart contracts. Automated checks were performed with Slither.

To standardise the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology.

Overall Risk Severity				
Impact	high	**MEDIUM**	**HIGH**	**CRITICAL**
	medium	**LOW**	**MEDIUM**	**HIGH**
	low	**NOTE**	**LOW**	**MEDIUM**
		low	medium	high
	Likelyhood			

- **Likelihood** represents how likely a particular vulnerability is to be uncovered and exploited in production
- **Impact** measures the business damage and technical loss of a successful attack
- **Severity** demonstrates the overall criticality of the risk

Likelihood and Impact are categorised into three ratings: high, medium and low. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e. **CRITICAL**, **HIGH**, **MEDIUM**, **LOW** and **NOTE** shown in table above.

To evaluate the risk, we go through a checklist of items and each would be labeled with a severity category.

We perform the audit according to the following procedure:

Configuration and basic coding bugs: study the source code and analyze given contracts with static code analysers and verify results manually.	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Black hole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	Use of Untrusted Libraries
	Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic consistency checks: manually check the business logic and compare with the specifications and description documents	Semantic Consistency Checks
Advanced DeFi Scrutiny: we review business logics, examine operations of the system and place DeFi-related aspects under scrutiny to uncover possible pitfalls and bugs	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow

	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional recommendations: we also identify and provide suggestions regarding the coding and development from the perspective of proven programming practices	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

To better describe each issue we identified, we categorised the findings with Smart Contract Weakness Classification and Test Cases Registry (SWC Registry) and Common Weakness Enumeration CWE-699, which is a community-developed list of software weakness types. Common Weakness Enumeration (CWE) Classifications Used in This Audit are:

Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management.
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.

Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an ex pilotable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

Findings summary

Here is a summary of our findings after analyzing the Metarun BEP-20 token contract implementation. Overall, this smart contract is well-designed and engineered, though we have so far identified the following potential issues:

Severity	№ of issues
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	0
NOTE	3
TOTAL	3

All the findings were discussed with the Metarun team and were recognized as safe, not requiring any actions. We believe this token passes security qualifications to be listed on digital assets exchanges.

Contract overview

The Token Solidity contract is located in a common repository along with other contracts that are still in progress of development so excluded from the audit scope.

For ERC-20 token functions, including Cap and Burn features, implementers used well-known and proven OpenZeppelin ERC-20 library. The token implements dynamic (conditional) minting required for how the “” works so this minter role will be revoked later on by the deployer after giving role to “rewards contract”.

To allow conditional minting and deflation mechanics, the token has mint() function that allows new supply issuance on demand (within the maximum cap). Authorization is implemented by means of OpenZeppelin’s Access-Control library. MINTER_ROLE is assigned to the deployer’s address on contract initialization and can be revoked and delegated to the contract responsible for rewards later.

All the libraries were imported and used in strict accordance with the OpenZeppelin’s recommendations and provided examples. Compliance with specifications verified by formal Hardhat tests located in test/MetarunToken.test.js (see the output in the Artifacts section below).

Slither static analyzer has detected 27 results, all are related to the usage of public vs external function modifiers in OpenZeppelin’s library, all concern only optimization only and don’t affect security so can be safely ignored.

Detected issues details

CRITICAL

No issue found

HIGH

No issue found

MEDIUM

No issue found

LOW

No issue found

NOTE

MRN-1 Old compiler pragma

Target: MetarunToken.sol:2, **Category**: Configuration, **SWC ID**: SWC-102

The contract has the following pragma directive:

```
1   //SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.0;
```

Currently, the latest version of the official Solidity compiler is v0.8.13, but at the time of contract deployment (February 23, 2022), the actual version was v0.8.11. As usual, it is recommended to stay on top of the new versions, which could avoid problems. But in the given case, later releases v0.8.12 and v0.8.18 introduced just minor bugfixes and improvements and don't seem to significantly affect the code.

Recommendation: since the contract is already deployed, no action required

MRN-2 Outdated library dependency

Target: yarn.lock:664 **Category**: Configuration

Currently deployed contracts use OpenZeppelin library of version v4.4.2 (actual a the contract deployment date)

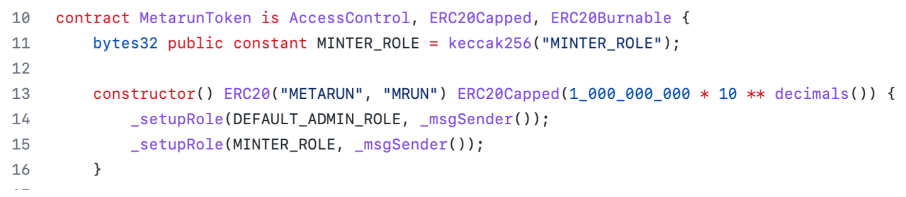
```
664  "@openzeppelin/contracts@^4.4.2":
665    version "4.4.2"
666    resolved "https://registry.yarnpkg.com/@openzeppelin/contracts/-/contracts-4.4.2.tgz#4e889c9c66e736f7de189a53f8ba5b8d789425c2"
667    integrity sha512-NyJV7sJgoGYqbtNUWgzzOGW4T6rR19FmX1IJgXGdapGPWsuMelGJn9h03nos0iqfforCbCB0iYIR0MtIuIFLLw==
```

Currently, the latest stable version of the OpenZeppelin is v4.5.0. If the contract had not been deployed, we would recommend updating the dependencies and “freezing” them with yarn. But for the reasons stated above in MRN-1, you don't need to touch this dependency.

Recommendation: since the contract is already deployed, no action required

MRN-3 Excessive authority of token deployer

Target: MetarunToken.sol:15 **Category**: Configuration



```
10  contract MetarunToken is AccessControl, ERC20Capped, ERC20Burnable {
11      bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
12
13      constructor() ERC20("METARUN", "MRUN") ERC20Capped(1_000_000_000 * 10 ** decimals()) {
14          _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
15          _setupRole(MINTER_ROLE, _msgSender());
16      }
```

The contract implements two roles - MINTER_ROLE and DEFAULT_ADMIN_ROLE that are assigned simultaneously by the constructor and both powers still present on the deployer address. Within the meaning of separation of powers, unnecessary capabilities of MINTER_ROLE should be revoked once supply was minted.

Note: Regardless of the presence/absence of MINTER_ROLE on the deployer’s address, the token is still capped at 1,000,000,000 units.

Recommendation: If the manual issuance of new tokens is not planned anymore, the MINTER_ROLE can be safely revoked and assigned to Reward contract then.

Disclaimer

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible. The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

Artifacts

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

Contract summary

Below is the contract summary plotted by Slither:

```
+ Contract AccessControl
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)
  - From AccessControl
    - _checkRole(bytes32,address) (internal)
    - _grantRole(bytes32,address) (internal)
    - _revokeRole(bytes32,address) (internal)
    - _setRoleAdmin(bytes32,bytes32) (internal)
    - _setupRole(bytes32,address) (internal)
    - getRoleAdmin(bytes32) (public)
    - grantRole(bytes32,address) (public)
    - hasRole(bytes32,address) (public)
    - renounceRole(bytes32,address) (public)
    - revokeRole(bytes32,address) (public)
    - supportsInterface(bytes4) (public)

+ Contract IAccessControl
  - From IAccessControl
    - getRoleAdmin(bytes32) (external)
    - grantRole(bytes32,address) (external)
    - hasRole(bytes32,address) (external)
    - renounceRole(bytes32,address) (external)
```

```
   - revokeRole(bytes32,address) (external)

+ Contract ERC20
  - From Context
   - _msgData() (internal)
   - _msgSender() (internal)
  - From ERC20
   - _afterTokenTransfer(address,address,uint256) (internal)
   - _approve(address,address,uint256) (internal)
   - _beforeTokenTransfer(address,address,uint256) (internal)
   - _burn(address,uint256) (internal)
   - _mint(address,uint256) (internal)
   - _transfer(address,address,uint256) (internal)
   - allowance(address,address) (public)
   - approve(address,uint256) (public)
   - balanceOf(address) (public)
   - constructor(string,string) (public)
   - decimals() (public)
   - decreaseAllowance(address,uint256) (public)
   - increaseAllowance(address,uint256) (public)
   - name() (public)
   - symbol() (public)
   - totalSupply() (public)
   - transfer(address,uint256) (public)
   - transferFrom(address,address,uint256) (public)

+ Contract IERC20
  - From IERC20
   - allowance(address,address) (external)
   - approve(address,uint256) (external)
   - balanceOf(address) (external)
   - totalSupply() (external)
   - transfer(address,uint256) (external)
   - transferFrom(address,address,uint256) (external)

+ Contract ERC20Burnable
  - From ERC20
   - _afterTokenTransfer(address,address,uint256) (internal)
   - _approve(address,address,uint256) (internal)
   - _beforeTokenTransfer(address,address,uint256) (internal)
   - _burn(address,uint256) (internal)
   - _mint(address,uint256) (internal)
   - _transfer(address,address,uint256) (internal)
   - allowance(address,address) (public)
   - approve(address,uint256) (public)
   - balanceOf(address) (public)
   - constructor(string,string) (public)
   - decimals() (public)
   - decreaseAllowance(address,uint256) (public)
   - increaseAllowance(address,uint256) (public)
   - name() (public)
   - symbol() (public)
   - totalSupply() (public)
   - transfer(address,uint256) (public)
   - transferFrom(address,address,uint256) (public)
  - From Context
   - _msgData() (internal)
   - _msgSender() (internal)
  - From ERC20Burnable
   - burn(uint256) (public)
   - burnFrom(address,uint256) (public)

+ Contract ERC20Capped
  - From ERC20
   - _afterTokenTransfer(address,address,uint256) (internal)
   - _approve(address,address,uint256) (internal)
   - _beforeTokenTransfer(address,address,uint256) (internal)
   - _burn(address,uint256) (internal)
   - _transfer(address,address,uint256) (internal)
   - allowance(address,address) (public)
   - approve(address,uint256) (public)
   - balanceOf(address) (public)
   - constructor(string,string) (public)
   - decimals() (public)
   - decreaseAllowance(address,uint256) (public)
   - increaseAllowance(address,uint256) (public)
   - name() (public)
   - symbol() (public)
   - totalSupply() (public)
   - transfer(address,uint256) (public)
   - transferFrom(address,address,uint256) (public)
  - From Context
   - _msgData() (internal)
   - _msgSender() (internal)
  - From ERC20Capped
   - _mint(address,uint256) (internal)
   - cap() (public)
   - constructor(uint256) (internal)

+ Contract IERC20Metadata
  - From IERC20
   - allowance(address,address) (external)
   - approve(address,uint256) (external)
   - balanceOf(address) (external)
   - totalSupply() (external)
   - transfer(address,uint256) (external)
   - transferFrom(address,address,uint256) (external)
  - From IERC20Metadata
   - decimals() (external)
   - name() (external)
   - symbol() (external)

+ Contract Context
  - From Context
   - _msgData() (internal)
   - _msgSender() (internal)

+ Contract Strings (Most derived contract)
  - From Strings
   - toHexString(uint256) (internal)
   - toHexString(uint256,uint256) (internal)
   - toString(uint256) (internal)

+ Contract ERC165
```

```
 - From ERC165
   - supportsInterface(bytes4) (public)

+ Contract IERC165
 - From IERC165
   - supportsInterface(bytes4) (external)

+ Contract MetarunToken (Most derived contract)
 - From ERC20Burnable
   - burn(uint256) (public)
   - burnFrom(address,uint256) (public)
 - From ERC20
   - _afterTokenTransfer(address,address,uint256) (internal)
   - _approve(address,address,uint256) (internal)
   - _beforeTokenTransfer(address,address,uint256) (internal)
   - _burn(address,uint256) (internal)
   - _transfer(address,address,uint256) (internal)
   - allowance(address,address) (public)
   - approve(address,uint256) (public)
   - balanceOf(address) (public)
   - constructor(string,string) (public)
   - decimals() (public)
   - decreaseAllowance(address,uint256) (public)
   - increaseAllowance(address,uint256) (public)
   - name() (public)
   - symbol() (public)
   - totalSupply() (public)
   - transfer(address,uint256) (public)
   - transferFrom(address,address,uint256) (public)
 - From Context
   - _msgData() (internal)
   - _msgSender() (internal)
 - From ERC20Capped
   - cap() (public)
   - constructor(uint256) (internal)
 - From AccessControl
   - _checkRole(bytes32,address) (internal)
   - _grantRole(bytes32,address) (internal)
   - _revokeRole(bytes32,address) (internal)
   - _setRoleAdmin(bytes32,bytes32) (internal)
   - _setupRole(bytes32,address) (internal)
   - getRoleAdmin(bytes32) (public)
   - grantRole(bytes32,address) (public)
   - hasRole(bytes32,address) (public)
   - renounceRole(bytes32,address) (public)
   - revokeRole(bytes32,address) (public)
   - supportsInterface(bytes4) (public)
 - From MetarunToken
   - _mint(address,uint256) (internal)
   - constructor() (public)
   - mint(address,uint256) (public)
```

Test suite output

Below is the output generated by running the hardhat test suite provided with the repository:

```
 Metarun token collection
   ✓ is deployed (975ms)
   Token creation
     ✓ should create character token (39ms)
     ✓ should create pet token
     ✓ should create artifact token (39ms)
     ✓ should create skin token
     ✓ should give zero if token does not exist
   Token transfer
     ✓ should transfer character token (43ms)
     ✓ should transfer pet token (39ms)
     ✓ should transfer artifact token
     ✓ should transfer skin token (38ms)
     ✓ should revert transfer of non-existing token (54ms)
   Token uri
     ✓ should correctly give uri for tokens
   Token roles
     ✓ should deny minting for non-minter (59ms)
     ✓ should perform minting for minter
     ✓ should deny changing uri for non-admin
     ✓ should perform changing uri for admin

<irrelevant output omitted>

 Metarun token
   ✓ has a name
   ✓ has a symbol
   ✓ has 18 decimals
   ✓ returns the total amount of tokens
   ✓ returns the balance of deployer
   ✓ user can burn his tokens
   ✓ unable to burn more than balance
   ✓ unable to exceed the cap
   ✓ unable to mint without MINTER_ROLE
   Checking token methods
     Basic transfers
       ✓ emits event Transfer
       ✓ spender's balance decreased
       ✓ receiver's balance increased
       ✓ reverts if account1 has not enough balance
     Approve
       ✓ emits event Approval
       ✓ allowance to account1
       ✓ reverts if ZERO_ADDRESS
       transferFrom
         ✓ emits event Transfer
         ✓ emits event Approval (allowance decreased)
         ✓ deployer allowance to account1
         ✓ deployer balance after transfer
         ✓ check account2 balance after transfer
```

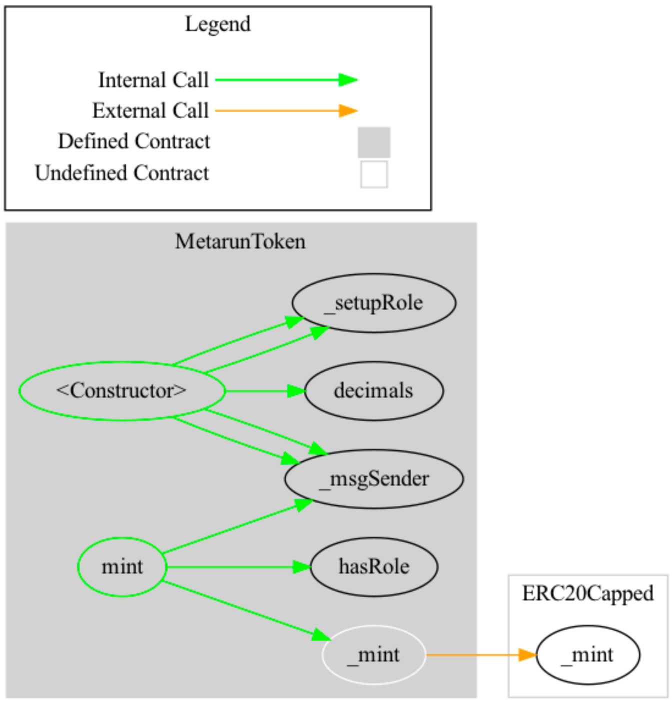
```
        ✓ reverts if account1 has not enough allowance
    increaseAllowance
      ✓ emits event Approval
      ✓ check deployer allowance to account1
      decreaseAllowance
        ✓ emits event Approval
        ✓ deployer allowance to account1
        ✓ reverts when decreased allowance below zero

<irrelevant output omitted>

88 passing (12s)
```

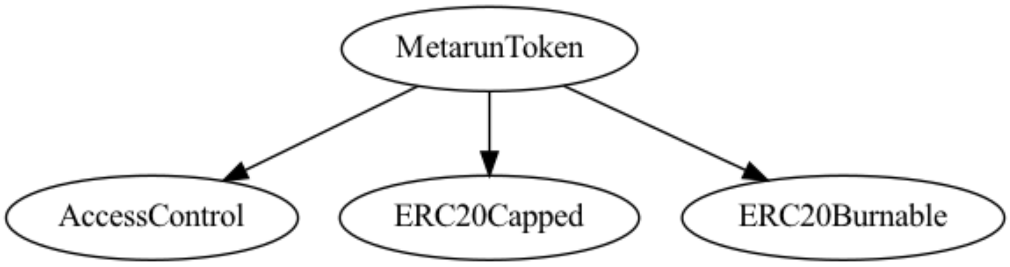
Call Graph

The following UML call flow diagram reflects internal end external calls of contract's functions. It's plotted using Surya utility tool.



Inheritance graph

The inheritance structure of the token contract (plotted with Surya)



Static analyser output

Output collected with Slither (irrelevant output was removed).

```
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/AccessControl.-
sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/IAccessControl.-
sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/
ERC20Burnable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/
ERC20Capped.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/
IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) al-
lows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows
old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ER-
C165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IER-
C165.sol#4) allows old versions
Pragma version^0.8.0 (contracts/MetarunToken.sol#2) allows old versions
solc-0.8.11 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-ver-
sions-of-solidity

grantRole(bytes32,address) should be declared external:
	- AccessControl.grantRole(bytes32,address) (node_modules/@openzeppelin/con-
tracts/access/AccessControl.sol#130-132)
revokeRole(bytes32,address) should be declared external:
	- AccessControl.revokeRole(bytes32,address) (node_modules/@openzeppelin/con-
tracts/access/AccessControl.sol#143-145)
renounceRole(bytes32,address) should be declared external:
	- AccessControl.renounceRole(bytes32,address) (node_modules/@openzeppelin/con-
tracts/access/AccessControl.sol#161-165)
name() should be declared external:
	- ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/
ERC20.sol#62-64)
symbol() should be declared external:
	- ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/
ERC20.sol#70-72)
balanceOf(address) should be declared external:
	- ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/
ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
	- ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/
ERC20/ERC20.sol#113-116)
approve(address,uint256) should be declared external:
	- ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/
ERC20/ERC20.sol#132-135)
transferFrom(address,address,uint256) should be declared external:
	- ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/con-
tracts/token/ERC20/ERC20.sol#150-164)
increaseAllowance(address,uint256) should be declared external:
	- ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/con-
tracts/token/ERC20/ERC20.sol#178-181)
decreaseAllowance(address,uint256) should be declared external:
	- ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/con-
tracts/token/ERC20/ERC20.sol#197-205)
burn(uint256) should be declared external:
	- ERC20Burnable.burn(uint256) (node_modules/@openzeppelin/contracts/token/ER-
C20/extensions/ERC20Burnable.sol#20-22)
burnFrom(address,uint256) should be declared external:
	- ERC20Burnable.burnFrom(address,uint256) (node_modules/@openzeppelin/con-
tracts/token/ERC20/extensions/ERC20Burnable.sol#35-42)
mint(address,uint256) should be declared external:
	- MetarunToken.mint(address,uint256) (contracts/MetarunToken.sol#32-38)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-func-
tion-that-could-be-declared-external
. analyzed (12 contracts with 77 detectors), 27 result(s) found
```