Cyberscope

Audit Report

MMIT burning contract

April 2023

Network BSC

Address 0x8C299e4320d902944075d0DEf2965d3fB804f759

Audited by © cyberscope

Table of Contents

Review

Explorer	https://bscscan.com/address/0x8c299e4320d902944075d0def2965d3fb804f759

Audit Updates

Initial Audit	30 Apr 2023

Source Files

Filename	SHA256
MMITBURN.sol	2f2d9bcb88e016e6762ce9a2328802b71d485af22913dadda35ac23ada133ef1

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

Introduction

The MMITBURN contract allows the owner to burn tokens periodically or on demand. The contract uses the SafeMath library for arithmetic operations and the IERC20 interface to interact with the token contract. Additionaly, any tokens transferred to this contract are unclaimable and can only be burned.

The `burnTimechange` function allows the owner to change the burnInterval variable.

The `startBurn` function allows the owner to burn 1% of the token balance if the last burn was at least burnInterval seconds ago. It checks if there are enough tokens to burn and transfers the tokens to the zero address (which is a way to burn tokens).

The `Burn` function allows the owner to burn a specific amount of tokens. It checks if there are enough tokens to burn and transfers the tokens to the zero address.

The `TokensBurned` event is emitted whenever tokens are burned.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	TUU	Time Units Usage	Unresolved
	RZAB	Redundant Zero Amount Burn	Unresolved
	CR	Code Repetition	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

TUU - Time Units Usage

Criticality	Minor / Informative
Location	MMITBURN.sol#L301
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values for the variable `burnInterval`. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 public burnInterval = 7776000 seconds;// 90 days interval;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days`, `weeks`, and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

RZAB - Redundant Zero Amount Burn

Criticality	Minor / Informative
Location	MMITBURN.sol#L315
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract `Burn` function performs the burning functionality even if zero amount is provided.

```
function Burn(uint256 amount) external onlyOwner {
    uint256 balance = token.balanceOf(address(this));
    require(amount <= balance, "Not Enough balance");
    token.transfer(0x000000000000000000000000000000000000dEaD, amount);
// Burn tokens by transferring to the zero address
    lastBurnTime = block.timestamp;
    emit TokensBurned(amount);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

The contract could integrate a sanity check on the burn amount. The burn amount should be greater than zero.

CR - Code Repetition

Criticality	Minor / Informative
Location	MMITBURN.sol#L315,324
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
require(burnAmount > 0, "No tokens available to burn.");
token.transfer(0x000000000000000000000000000000000000dEaD, burnAmount);
// Burn tokens by transferring to the zero address
lastBurnTime = block.timestamp;
emit TokensBurned(burnAmount);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MMITBURN.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases the gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MMITBURN.sol#L304,305
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
token
owner
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MMITBURN.sol#L320
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function Burn(uint256 amount) external onlyOwner {
        uint256 balance = token.balanceOf(address(this));
        require(amount <= balance, "Not Enough balance");

token.transfer(0x000000000000000000000000000000000000dEaD,
amount); // Burn tokens by transferring to the zero address
        lastBurnTime = block.timestamp;
        emit TokensBurned(amount);
    }
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MMITBURN.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	MMITBURN.sol#L316,323
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(0x000000000000000000000000000000000000dEaD,
burnAmount)
token.transfer(0x000000000000000000000000000000000000dEaD,
amount)
```

Recommendation

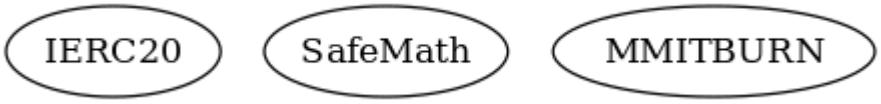
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

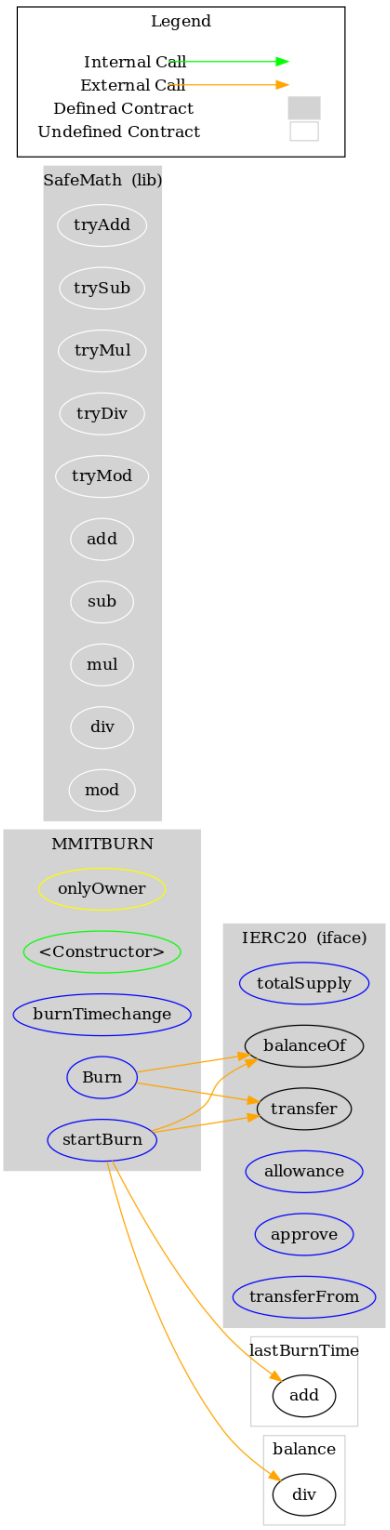
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
MMITBURN	Implementation			
		Public	✓	-
	burnTimechange	External	✓	onlyOwner
	startBurn	External	✓	onlyOwner
	Burn	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
MMITBURN
onlyOwner
<Constructor>
burnTimechange
Burn
startBurn
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
lastBurnTime
add
balance
div

Summary

MMITBURN contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io