Ether Authority

etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Magic Internet Money
Website:	abracadabra.money
Platform:	Base Chain Network
Language:	Solidity
Date:	June 18th, 2024

Table of contents

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Magic Internet Money smart contract from abracadabra.money was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 18th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.

- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Website Details

- Abracadabra.money is a Omnichain DeFi lending platform that works its magic by utilizing interest-bearing tokens as collateral to mint Magic Internet Money, a USD-Denominated stablecoin.
- It enables users to leverage interest-bearing tokens as collateral to mint the stablecoin MIM (Magic Internet Money). The platform supports various DeFi strategies and offers services like borrowing, staking, and farming to maximize yield.

Code Details

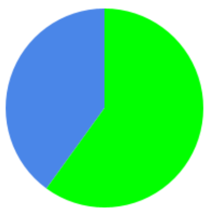
- This Solidity contract defines a Mintable and Burnable ERC20 token with owner and operator permissions.
- **Components:**
  - IMintableBurnable Interface: Defines burn and mint functions.
  - Owned Contract: Manages ownership with transfer capabilities.

    - ERC20 Contract: Implements ERC20 token functionalities and EIP-2612 permits.
    - OperatableV2 Contract: Manages operators who can mint and burn tokens.
    - MintableBurnableERC20 Contract: Combines the above contracts, allowing only operators to mint and burn tokens.
- **Key Functions:**
    - mint(address to, uint256 amount)`: Mints new tokens to a specified address.
    - burn(address from, uint256 amount)`: Burns tokens from a specified address.
    - setOperator(address operator, bool status)`: Sets an address as an operator.

Audit scope

Name	**Code Review and Security Analysis Report for Magic Internet Money(MIM) Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	MintableBurnableERC20.sol
Smart Contract Code	0x4A3A6Dd60A34bB2Aba60D73B4C88315E9CeB6A3D
Audit Date	June 18th,2024
Audit Result	**Passed**

Code Audit History



5	0	0	0	3	2
Total Findings	Critical	High	Medium	Low	Informational

Severity Definitions

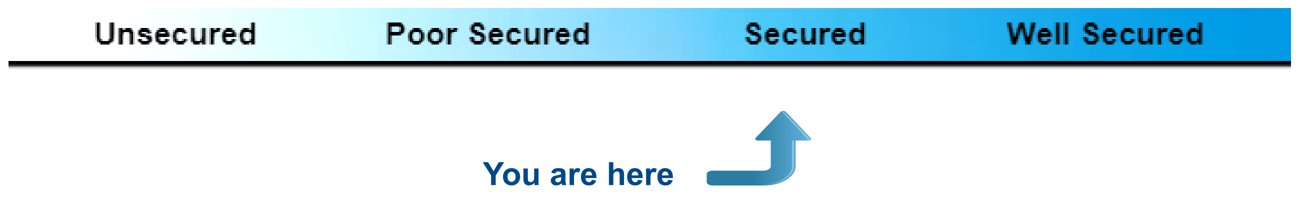
0 🟥	**Critical**		Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0 🟧	**High**		High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0 🟨	**Medium**		Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
3 🟩	**Low**	🟩 🟩 🟩	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
2 🟦	**Lowest / Informational / Best Practice**	🟦 🟦	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Magic Internet Money • Symbol: MIM • Decimals: 18	**YES, This is valid.**
Key Functions: • mint(address to, uint256 amount)`: Mints new tokens to a specified address. • burn(address from, uint256 amount)`: Burns tokens from a specified address. • setOperator(address operator, bool status)`: Sets an address as an operator.	**YES, This is valid.**
Owner Specifications: • The operator's address can be updated. • Allows the current owner to transfer control of the contract to a new owner.	**YES, This is valid.** **We advise renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**.Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 3 low, and 2 very low-level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	No
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	Yes
● Is it a Proxy Contract?	No
● Is it used Open Source?	No
● External Call Risk?	No
● Balance Modifiable?	No
● Can Take Ownership?	Yes
● Ownership Renounce?	No
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Magic Internet Money are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Magic Internet Money.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Magic Internet Money smart contract code in the form of a basescan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

MintableBurnableERC20.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Empty Constructor Present	Refer Audit Findings
2	burn	external	Operators can burn anyone's token	Refer Audit Findings
3	mint	external	Unlimited token minting	Refer Audit Findings
4	onlyOperators	modifier	Passed	No Issue
5	setOperator	external	access only owner	No Issue
6	approve	write	Passed	No Issue
7	transfer	write	Passed	No Issue
8	transferFrom	write	Passed	No Issue
9	permit	write	Passed	No Issue
10	DOMAIN_SEPARATOR	read	Passed	No Issue
11	computeDomainSeparator	internal	Passed	No Issue
12	_mint	internal	Passed	No Issue
13	_burn	internal	Passed	No Issue
14	onlyOwner	modifier	Passed	No Issue
15	transferOwnership	write	Function input parameters lack of check	Refer Audit Findings

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

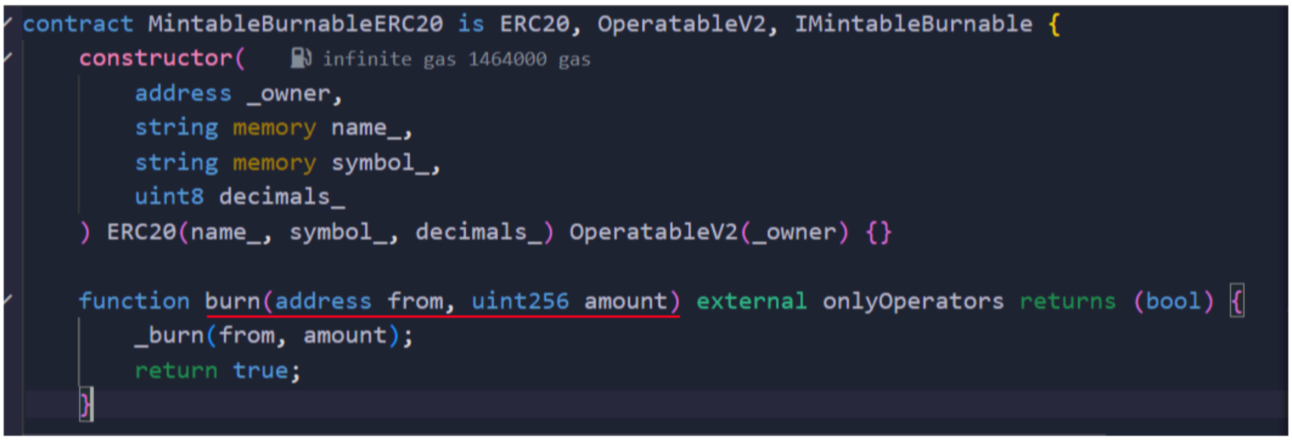
No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

[L-01] Operators can burn anyone’s token:



```
contract MintableBurnableERC20 is ERC20, OperatableV2, IMintableBurnable {
    constructor(    infinite gas 1464000 gas
        address _owner,
        string memory name_,
        string memory symbol_,
        uint8 decimals_
    ) ERC20(name_, symbol_, decimals_) OperatableV2(_owner) {}

    function burn(address from, uint256 amount) external onlyOperators returns (bool) {
        _burn(from, amount);
        return true;
    }
```

Description:

Operators can burn any user’s tokens.

Recommendation: We suggest changing the code so only token holders can burn their own tokens and not anyone else. Not even a contract creator.

[L-02] Unlimited token minting:

```
 function mint(address to, uint256 amount) external onlyOperators
returns (bool) {
        _mint(to, amount);
        return true;
    }
```

Description:

The contract allows unlimited token creation, risking inflation and devaluation.

Recommendation: We suggest Implementing a maximum token supply cap and restricting minting permissions to trusted entities.

[L-03] Function input parameters lack of check:

```
 function transferOwnership(address newOwner) public virtual
onlyOwner {
        owner = newOwner;

        emit OwnershipTransferred(msg.sender, newOwner);
    }
```

Description:

functions require validation before execution.

Functions are:

- transferOwnership()

Recommendation: We suggest using validation, like for numerical variables that should be greater than 0, and for address-type check variables that are not addressed (0). For percentage-type variables, values should have some range, like a minimum of 0 and a maximum of 100.

Very Low / Informational / Best practices:

[I-01] Empty Constructor Present:

```
   constructor(
        address _owner,
        string memory name_,
        string memory symbol_,
        uint8 decimals_
    ) ERC20(name_, symbol_, decimals_) OperatableV2(_owner) {}
```

Description:
The smart contract includes an empty constructor, which serves no functional purpose. This can lead to confusion and unnecessary code complexity.

Recommendation: We suggest considering removing the empty constructor to clean up the code. If the constructor is intended for future use, ensure it includes relevant logic or documentation to clarify its purpose.

[I-02] Eliminate IMintableBurnable Interface Dependency:

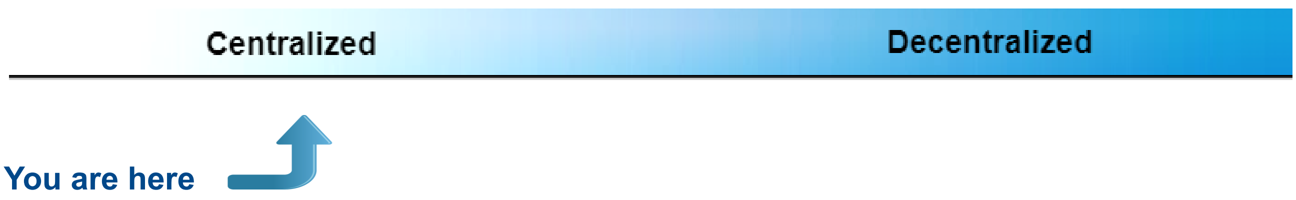
```
interface IMintableBurnable {
    function burn(address from, uint256 amount) external returns
(bool);

    function mint(address to, uint256 amount) external returns
(bool);
}
```

Description:
If the IMintableBurnable interface is removed, it will not affect the existing functionality.

Recommendation: We suggest IMintableBurnable interface, Removing the IMintableBurnable interface would not affect the functionality of your contract directly because the actual implementation of the burn and mint functions exists within the MintableBurnableERC20 contract itself.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.



The following are Owner functions:

OperatableV2.sol

- setOperator: The operator address can be set by the owner.

Ownable.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a newOwner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a basescan web link. And we have used all possible tests based on given objects as files. We observed 3 low and 2 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

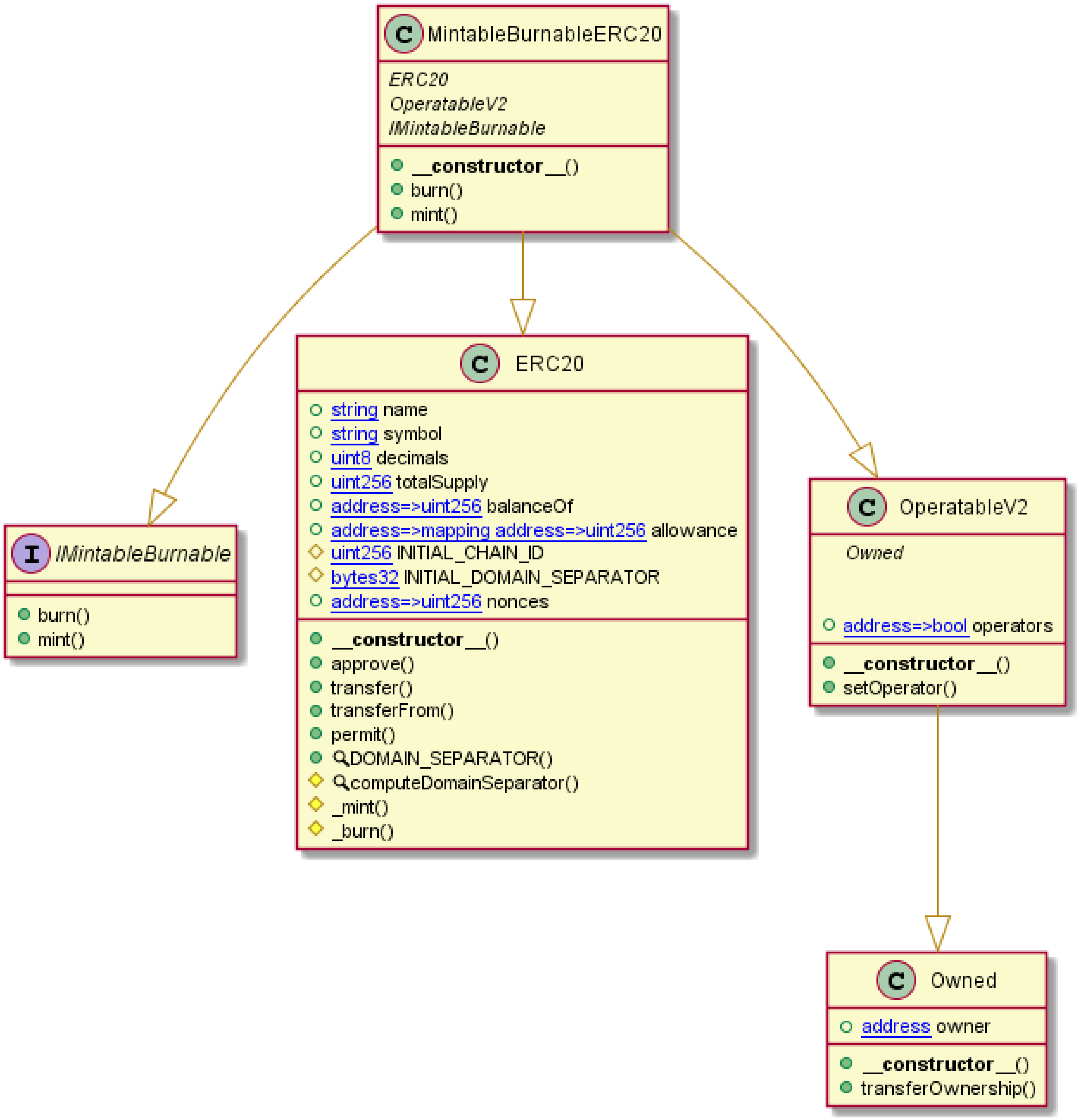
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Magic Internet Money



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

MintableBurnableERC20.sol

```
INFO:Detectors:
Owned.transferOwnership(address).newOwner (MintableBurnableERC20.sol#44) lacks a
zero-check on :
            - owner = newOwner (MintableBurnableERC20.sol#45)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32)
(MintableBurnableERC20.sol#160-204) uses timestamp for comparisons
      Dangerous comparisons:
      - require(bool,string)(deadline >= block.timestamp,PERMIT_DEADLINE_EXPIRED)
(MintableBurnableERC20.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Pragma version>=0.8.0 (MintableBurnableERC20.sol#2) allows old versions
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function ERC20.DOMAIN_SEPARATOR() (MintableBurnableERC20.sol#206-208) is not in
mixedCase
Variable ERC20.INITIAL_CHAIN_ID (MintableBurnableERC20.sol#85) is not in mixedCase
Variable ERC20.INITIAL_DOMAIN_SEPARATOR (MintableBurnableERC20.sol#87) is not in
mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:MintableBurnableERC20.sol analyzed (5 contracts with 93 detectors), 7 result(s)
found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

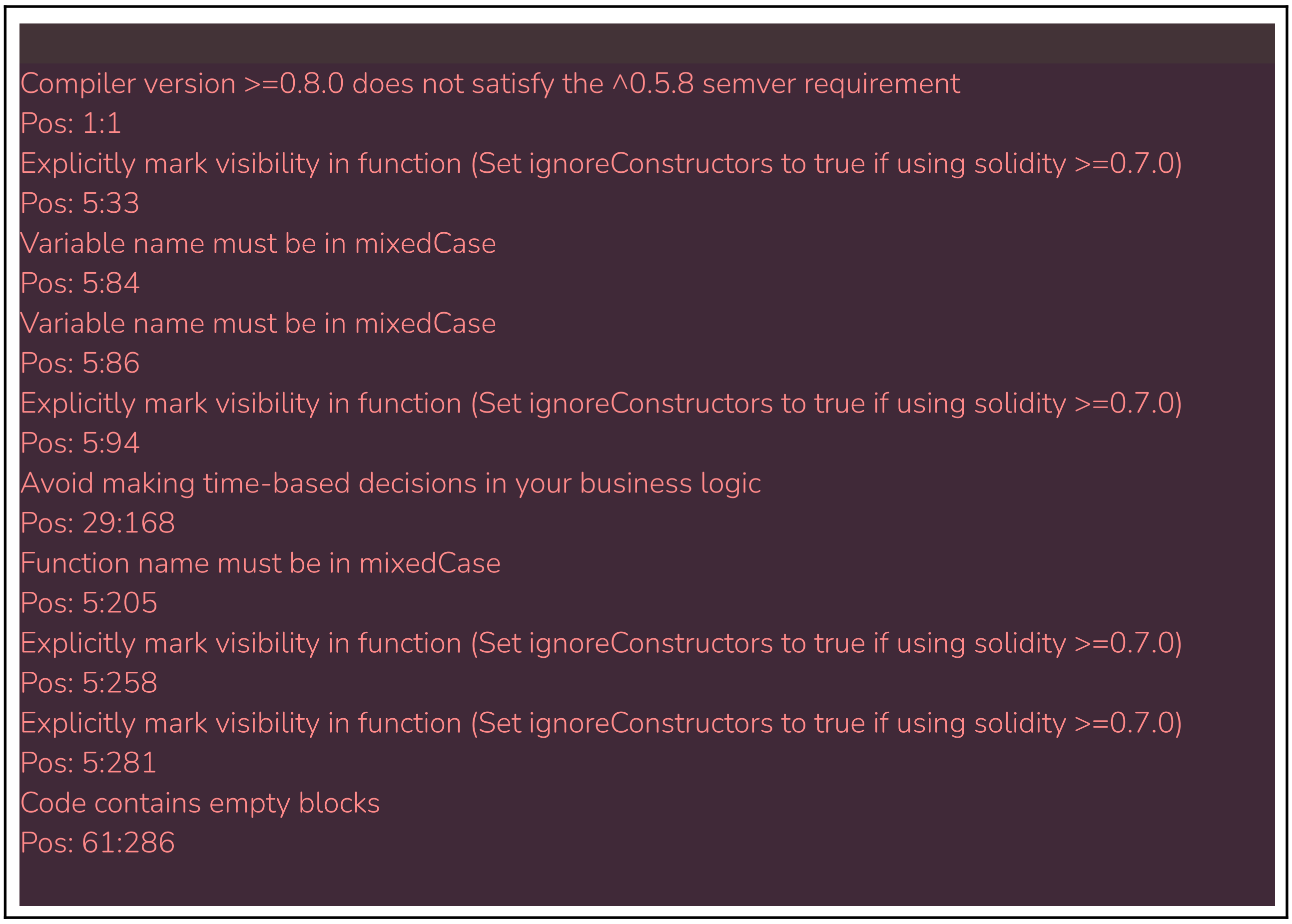
MintableBurnableERC20.sol

```
Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the
outcome of a transaction in the mined block.
Pos: 169:28:

Gas costs:
Gas requirement of function MintableBurnableERC20.mint is infinite: If the gas requirement of a
function is higher than the block gas limit, it cannot be executed. Please avoid loops in your
functions or actions that modify large areas of storage (this includes clearing or copying arrays in
storage)
Pos: 294:4:

Constant/View/Pure functions:
IMintableBurnable.burn(address,uint256) : Potentially should be constant/view/pure but is not.
Note: Modifiers are currently not considered by this static analysis.
Pos: 5:4:

Similar variable names:
OperatableV2.setOperator(address,bool) : Variables have very similar names "operator" and
"operators". Note: Modifiers are currently not considered by this static analysis.
Pos: 272:29:

No return:
IMintableBurnable.mint(address,uint256): Defines a return type but never explicitly returns a
value.
Pos: 7:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in
your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external
component.
Pos: 198:12:
```

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

MintableBurnableERC20.sol



```
Compiler version >=0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:33
Variable name must be in mixedCase
Pos: 5:84
Variable name must be in mixedCase
Pos: 5:86
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:94
Avoid making time-based decisions in your business logic
Pos: 29:168
Function name must be in mixedCase
Pos: 5:205
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:258
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:281
Code contains empty blocks
Pos: 61:286
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.