Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	USDM Token
Website:	Goldchainex.com, gcex.lt
Platform:	Binance Smart Chain
Language:	Solidity
Date:	June 8th, 2022

Table of contents

Introduction 4

Project Background 4

Audit Scope 4

Claimed Smart Contract Features 5

Audit Summary 6

Technical Quick Stats 7

Code Quality 8

Documentation 8

Use of Dependencies 8

AS-IS overview 9

Severity Definitions 11

Audit Findings 12

Conclusion 16

Our Methodology 17

Disclaimers 19

Appendix

- Unified Modeling Language (UML) 20
- Call Graph 21
- Automatic General Report 22
- Slither Results Log 25
- Solidity static analysis 26
- Solhint Linter 28

`

Introduction

EtherAuthority was contracted by the USDM team to perform the Security audit of the USDM smart contract code. 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 8th, 2022.

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

USDM token is a stable token which can be minted, burned through the minters added by the owner.

Audit scope

Name	**Code Review and Security Analysis Report for USDM Smart Contract**
Platform	**BSC / Solidity**
File	USDm.sol
File MD5 Hash	8DC0B0E39282A55FB7AB2B5D30C69C86
Updated File MD5 Hash	82334197B6A0421CD4B0DFB4CCD0D6DA
Online Code Link	https://github.com/kesaviwebsolutions/usdm-contracts/blob/main/USDm.sol
Audit Date	June 8th, 2022
Revise Audit Date	June 14th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Ownership Control: ● Master Minter can add/update/remove minter address. ● Minter can mint the allowed number of tokens.	YES, This is valid.

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Secured"**. This stable token contract does contain owner control, which does not make it fully decentralized, however, the smart contract provides features and ability to operate individual functions by independent and accredited third party to operate in a fully-decentralized environment with credibility as per the whitepaper.

Insecure
Poor secured
Secure
Well-secured
You are here

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 4 low and 0 very low level issues.

All the issues have been resolved in the revised code.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	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	Passed
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 not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

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

The libraries in USDM Token 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 USDM Token.

The USDM Token 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 USDM Token smart contract code in the form of a Github web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. 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.

Another source of information was its official websites http://Goldchainex.com and http://gcex.lt/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

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

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	rescuer	external	Passed	No Issue
8	onlyRescuer	modifier	Passed	No Issue
9	rescueERC20	external	access only Rescuer	No Issue
10	updateRescuer	external	access only Owner	No Issue
11	onlyPauser	modifier	Passed	No Issue
12	paused	read	Passed	No Issue
13	whenNotPaused	modifier	Passed	No Issue
14	whenPaused	modifier	Passed	No Issue
15	_pause	internal	Passed	No Issue
16	_unpause	internal	Passed	No Issue
17	_approve	internal	Passed	No Issue
18	_transfer	internal	Passed	No Issue
19	onlyBlacklister	modifier	Passed	No Issue
20	notBlacklisted	modifier	Passed	No Issue
21	isBlacklisted	external	Passed	No Issue
22	blacklist	external	access only Blacklister	No Issue
23	unBlacklist	external	access only Blacklister	No Issue
24	updateBlacklister	external	access only Owner	No Issue
25	initialize	write	Passed	No Issue
26	onlyMinters	modifier	Passed	No Issue
27	mint	external	access only onlyMinters	No Issue
28	onlyMasterMinter	modifier	Passed	No Issue
29	minterAllowance	external	Passed	No Issue
30	isMinter	external	Passed	No Issue
31	allowance	external	Passed	No Issue
32	totalSupply	external	Passed	No Issue
33	balanceOf	external	Passed	No Issue
34	approve	external	Passed	No Issue
35	_approve	internal	Passed	No Issue
36	transferFrom	external	Passed	No Issue
37	transfer	external	Passed	No Issue
38	_transfer	internal	Passed	No Issue

39	configureMinter	external	access only onlyMasterMinter	No Issue
40	removeMinter	external	access only onlyMasterMinter	No Issue
41	burn	external	Passed	No Issue
42	updateMasterMinter	external	access only Owner	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can’t affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

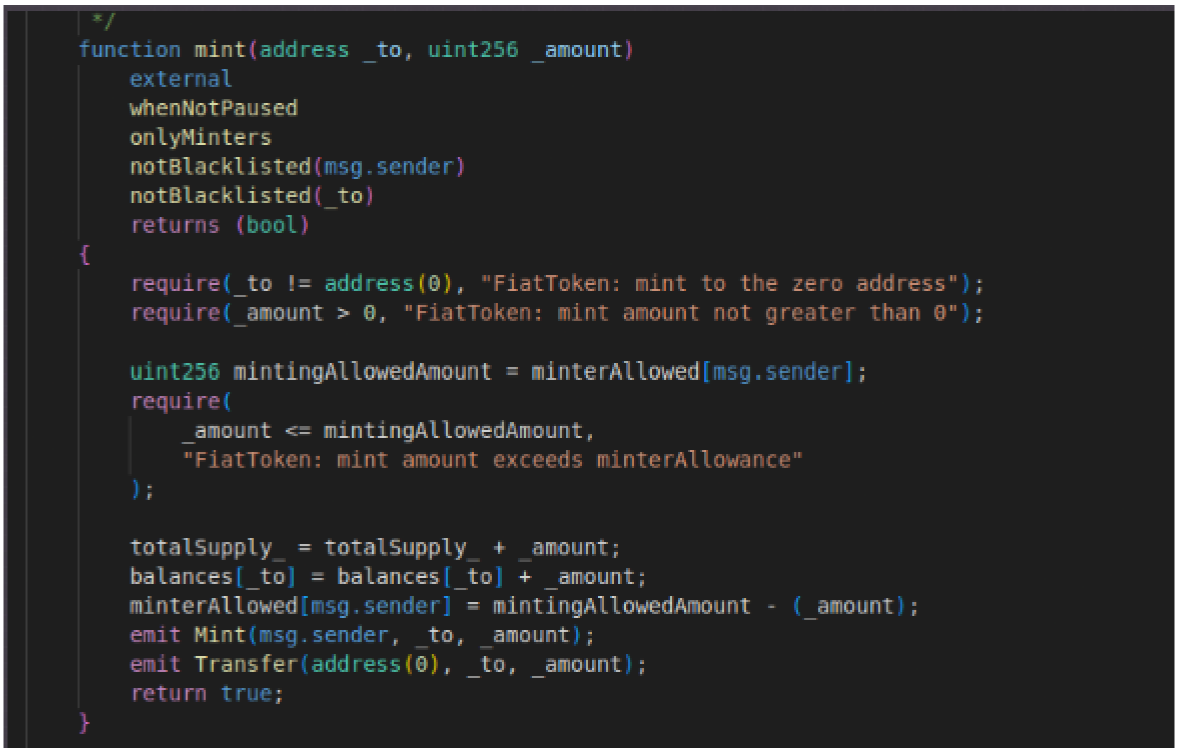
No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) No Max minting of the tokens set:



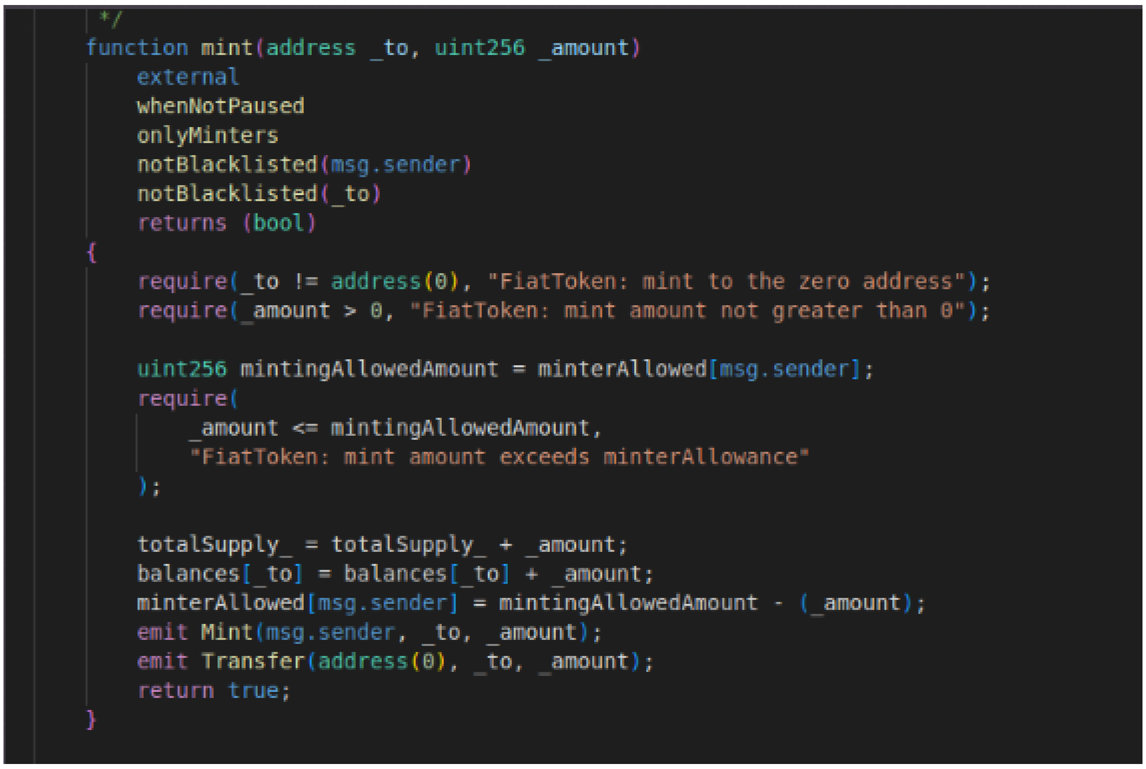
```
    */
    function mint(address _to, uint256 _amount)
        external
        whenNotPaused
        onlyMinters
        notBlacklisted(msg.sender)
        notBlacklisted(_to)
        returns (bool)
    {
        require(_to != address(0), "FiatToken: mint to the zero address");
        require(_amount > 0, "FiatToken: mint amount not greater than 0");

        uint256 mintingAllowedAmount = minterAllowed[msg.sender];
        require(
            _amount <= mintingAllowedAmount,
            "FiatToken: mint amount exceeds minterAllowance"
        );

        totalSupply_ = totalSupply_ + _amount;
        balances[_to] = balances[_to] + _amount;
        minterAllowed[msg.sender] = mintingAllowedAmount - (_amount);
        emit Mint(msg.sender, _to, _amount);
        emit Transfer(address(0), _to, _amount);
        return true;
    }
```

Setting max minting for the tokens is good for tokenomics.

Resolution: Since this is a stable coin smart contract, and thus it is necessary to have a minting feature. Thus, this should be taken care of from the client Side.

Status: Fixed.

(2) Owner and masterMinter can’t mint tokens for other users:

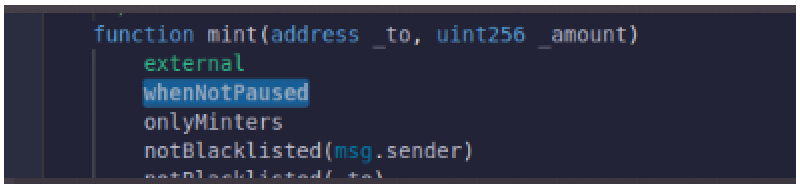

```
    */
    function mint(address _to, uint256 _amount)
        external
        whenNotPaused
        onlyMinters
        notBlacklisted(msg.sender)
        notBlacklisted(_to)
        returns (bool)
    {
        require(_to != address(0), "FiatToken: mint to the zero address");
        require(_amount > 0, "FiatToken: mint amount not greater than 0");

        uint256 mintingAllowedAmount = minterAllowed[msg.sender];
        require(
            _amount <= mintingAllowedAmount,
            "FiatToken: mint amount exceeds minterAllowance"
        );

        totalSupply_ = totalSupply_ + _amount;
        balances[_to] = balances[_to] + _amount;
        minterAllowed[msg.sender] = mintingAllowedAmount - (_amount);
        emit Mint(msg.sender, _to, _amount);
        emit Transfer(address(0), _to, _amount);
        return true;
    }
```

Owner and masterMinter can’t mint tokens from contract because mint function can be called by onlyMinters. Only minters can mint tokens for themselves.

Resolution: We suggest allowing the owner or masterMinter to mint USDm tokens for other users.

Status: Fixed.

(3) Unused pausable contract:

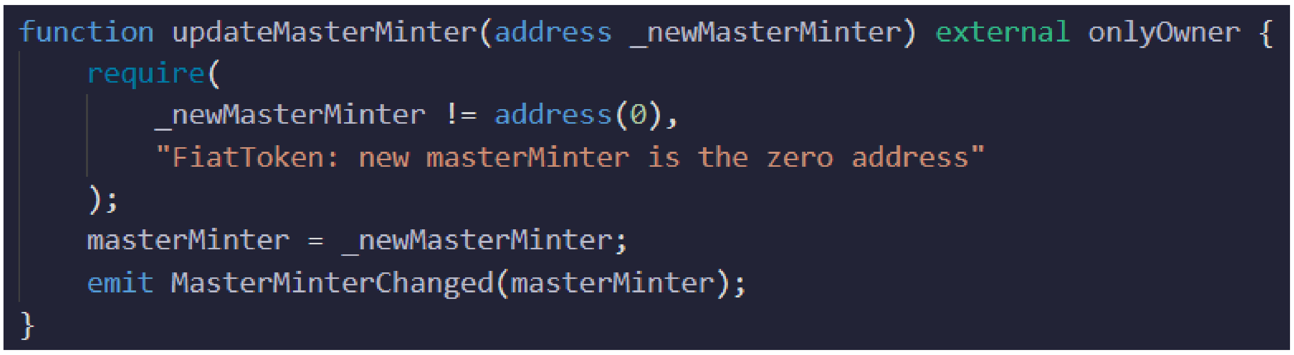

```
    function mint(address _to, uint256 _amount)
        external
        whenNotPaused
        onlyMinters
        notBlacklisted(msg.sender)
```

There is no function to pause or unpause the contract.

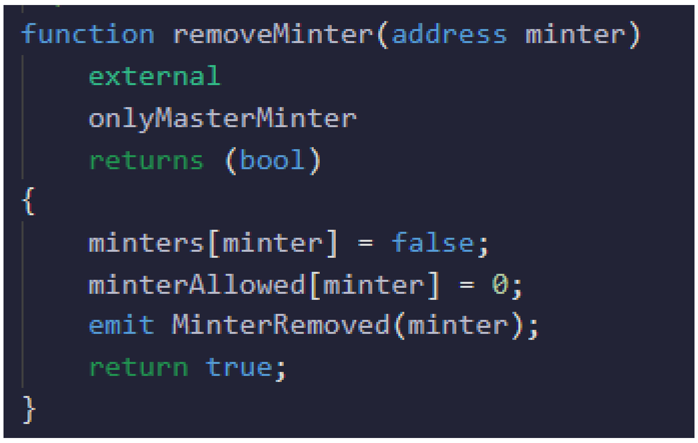
Resolution: Add pause and unpause owner functions in contract or remove the pausable functionality.

Status: Fixed.

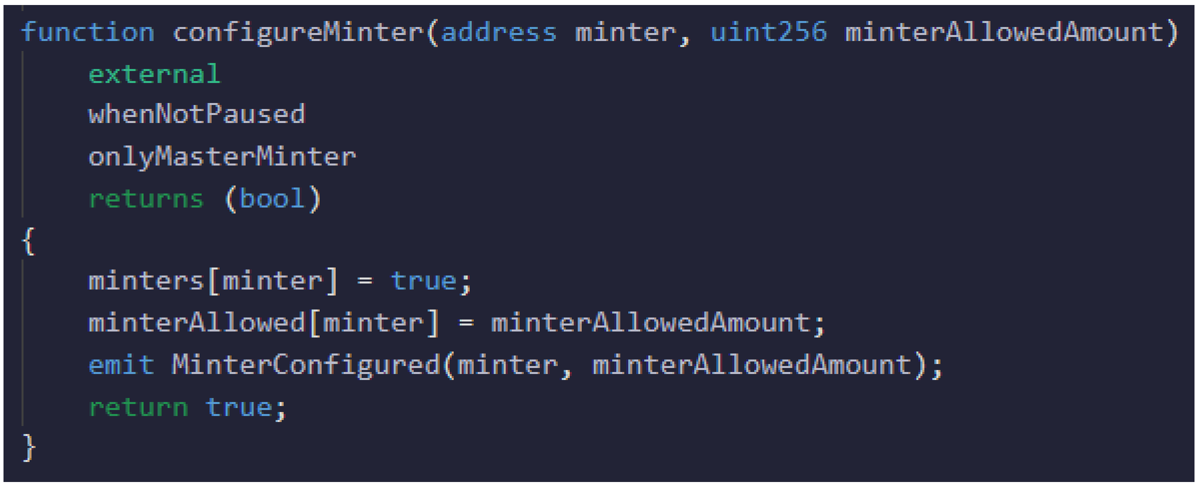
(4) Function execution resulting in unintended gas cost:



```
function updateMasterMinter(address _newMasterMinter) external onlyOwner {
    require(
        _newMasterMinter != address(0),
        "FiatToken: new masterMinter is the zero address"
    );
    masterMinter = _newMasterMinter;
    emit MasterMinterChanged(masterMinter);
}
```



```
function removeMinter(address minter)
    external
    onlyMasterMinter
    returns (bool)
{
    minters[minter] = false;
    minterAllowed[minter] = 0;
    emit MinterRemoved(minter);
    return true;
}
```



```
function configureMinter(address minter, uint256 minterAllowedAmount)
    external
    whenNotPaused
    onlyMasterMinter
    returns (bool)
{
    minters[minter] = true;
    minterAllowed[minter] = minterAllowedAmount;
    emit MinterConfigured(minter, minterAllowedAmount);
    return true;
}
```

Not checking if the current value is the same as the new value while updating masterMinters in updateMasterMinte(). This may lead to gas fee wastage if the function is called with the same value as current. The same also goes for removeMinter() and configureMinter().

Resolution: Include a check to see if the current value is the same as the new value.
Status: Fixed.

Very Low / Informational / Best practices:

No Informational severity vulnerabilities were found.

Centralization

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

- mint: Minter owner can mint tokens.
- configureMinter: Master Minter owner can add/update a new minter address.
- removeMinter: Master Minter owner can remove minter address.
- burn: Minter owner can burn some of its own tokens.
- updateMasterMinter: Owner can update new master minter address.

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 form of a Github web link. And we have used all possible tests based on given objects as files. We have not observed any major issues. So, **The smart contract is 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 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.

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

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 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 whitebox penetration testing. We look at the project's web site 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).

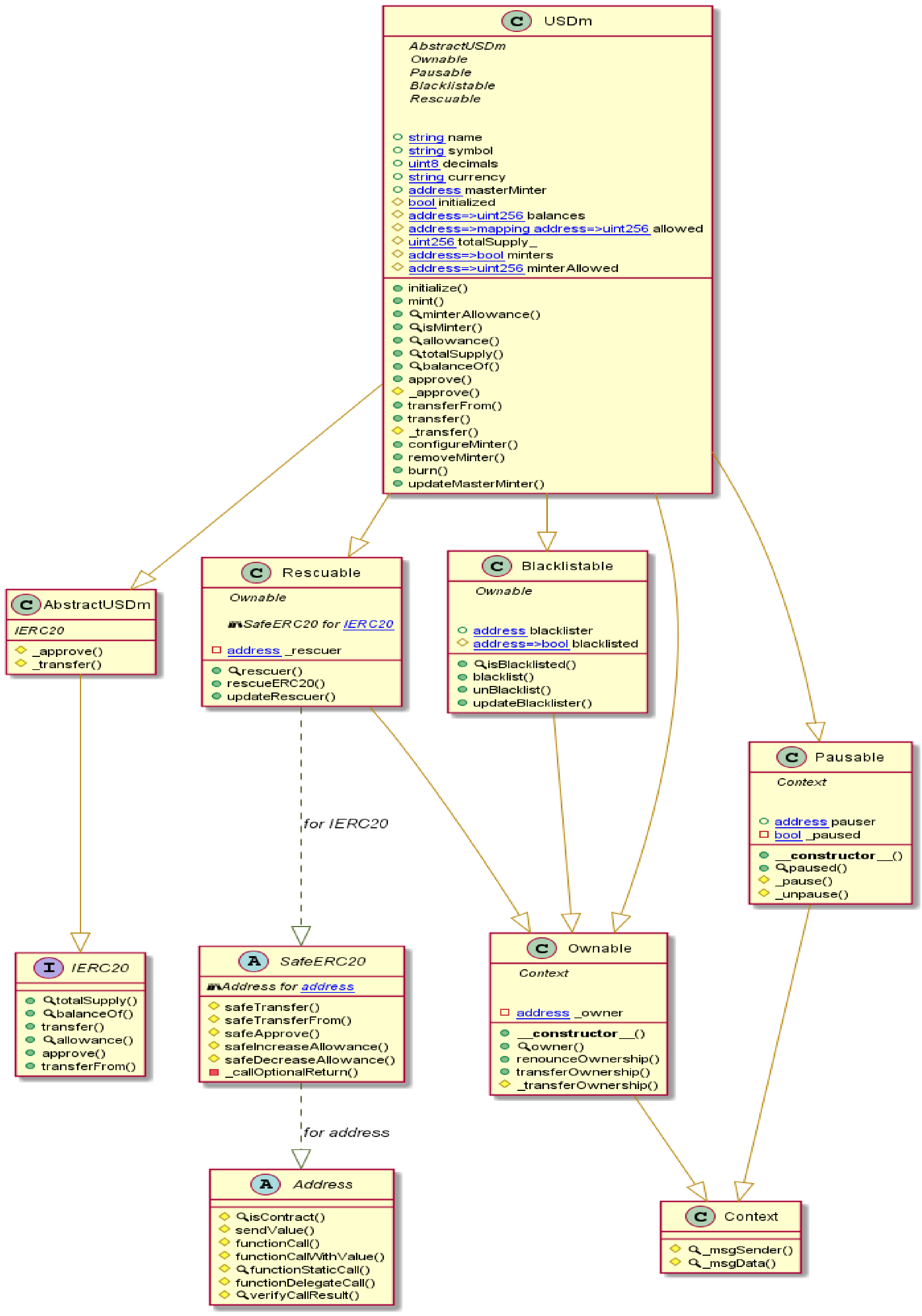
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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

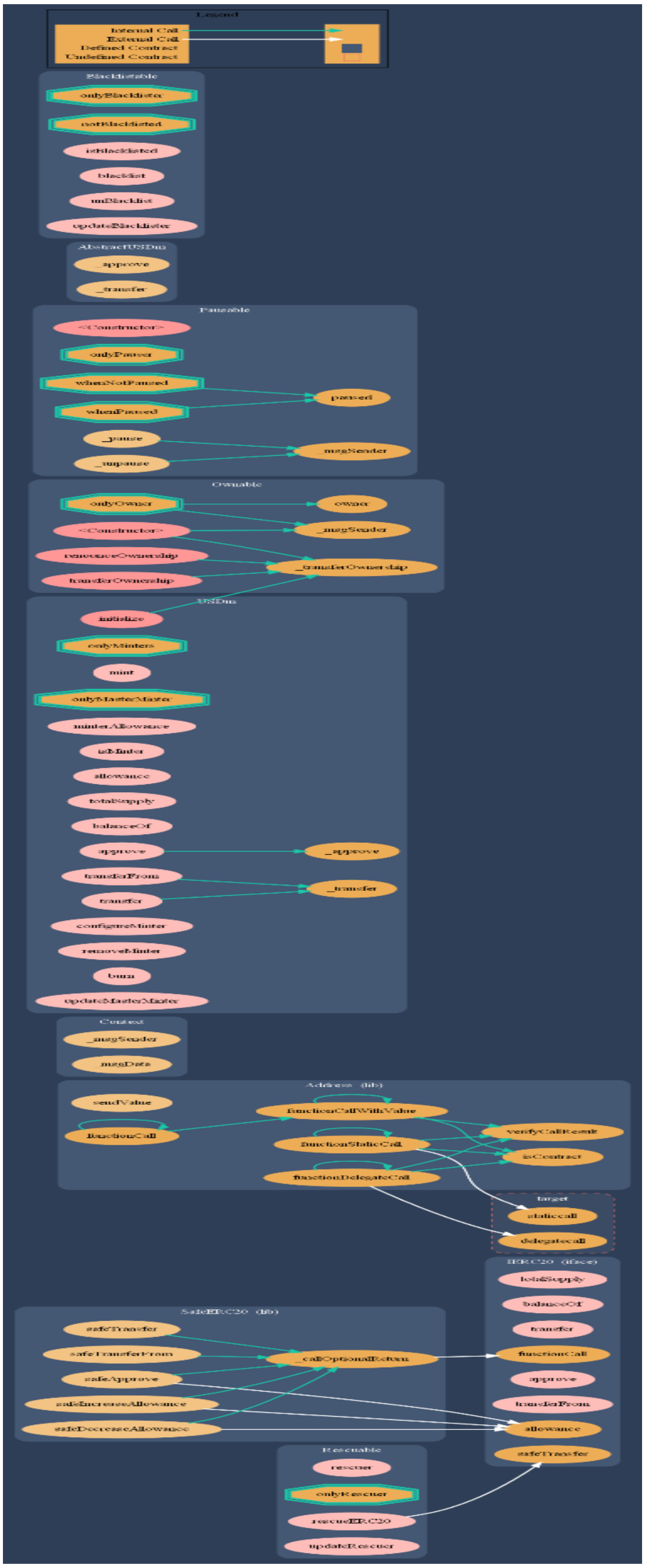
Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Unified Modeling Language (UML) - USDM Token



USDM - Call Graph



Automatic General Report

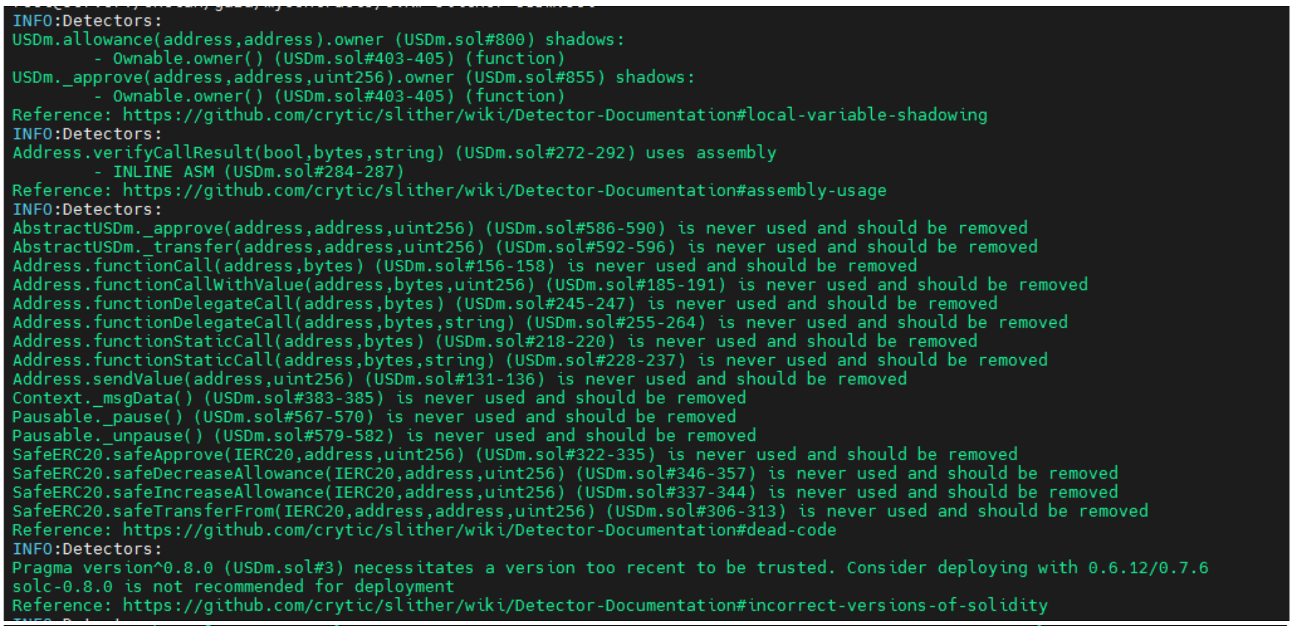
```
 Sūrya's Description Report

 Files Description Table


|  File Name  |  SHA-1 Hash  |
|-------------|--------------|
| e:\xampp\htdocs\solidity\USDm.sol |
d79f6d6b2137a0d3c8bd95af7bc7d7aa08f6faee |
| e:\xampp\htdocs\solidity\(14)
Dusktopia\DusktopiaAudit-master\src\Dusktopia.sol |
1a7c0795c2c7f9cf9cf0289c0bf1946aafba89a8 |
| e:\xampp\htdocs\solidity\USDm.sol |
d79f6d6b2137a0d3c8bd95af7bc7d7aa08f6faee |



 Contracts Description Table


|  Contract  |         Type        |       Bases      |                  |
|
|:----------:|:-------------------:|:----------------:|:----------------:|:-
---------------:|
|     └      |  **Function Name**  |  **Visibility**  |  **Mutability**  |
**Modifiers**  |
||||||
| **IERC20** | Interface |  |||
| └ | totalSupply | External ❗️ |   |NO❗️ |
| └ | balanceOf | External ❗️ |   |NO❗️ |
| └ | transfer | External ❗️ | 🛑  |NO❗️ |
| └ | allowance | External ❗️ |   |NO❗️ |
| └ | approve | External ❗️ | 🛑  |NO❗️ |
| └ | transferFrom | External ❗️ | 🛑  |NO❗️ |
||||||
| **Address** | Library |  |||
| └ | isContract | Internal 🔒 |   | |
| └ | sendValue | Internal 🔒 | 🛑  | |
| └ | functionCall | Internal 🔒 | 🛑  | |
| └ | functionCall | Internal 🔒 | 🛑  | |
| └ | functionCallWithValue | Internal 🔒 | 🛑  | |
| └ | functionCallWithValue | Internal 🔒 | 🛑  | |
| └ | functionStaticCall | Internal 🔒 |   | |
| └ | functionStaticCall | Internal 🔒 |   | |
| └ | functionDelegateCall | Internal 🔒 | 🛑  | |
| └ | functionDelegateCall | Internal 🔒 | 🛑  | |
| └ | verifyCallResult | Internal 🔒 |   | |
||||||
| **SafeERC20** | Library |  |||
| └ | safeTransfer | Internal 🔒 | 🛑  | |
| └ | safeTransferFrom | Internal 🔒 | 🛑  | |
| └ | safeApprove | Internal 🔒 | 🛑  | |
| └ | safeIncreaseAllowance | Internal 🔒 | 🛑  | |
| └ | safeDecreaseAllowance | Internal 🔒 | 🛑  | |
| └ | _callOptionalReturn | Private 🔐 | 🛑  | |
||||||
| **Context** | Implementation |  |||
| └ | _msgSender | Internal 🔒 |   | |
| └ | _msgData | Internal 🔒 |   | |
||||||
| **Ownable** | Implementation | Context |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | owner | Public ❗️ |   |NO❗️ |
| └ | renounceOwnership | Public ❗️ | 🛑  | onlyOwner |
| └ | transferOwnership | Public ❗️ | 🛑  | onlyOwner |
| └ | _transferOwnership | Internal 🔒 | 🛑  | |
||||||
| **Rescuable** | Implementation | Ownable |||
| └ | rescuer | External ❗️ |   |NO❗️ |
| └ | rescueERC20 | External ❗️ | 🛑  | onlyRescuer |
| └ | updateRescuer | External ❗️ | 🛑  | onlyOwner |
||||||
| **Pausable** | Implementation | Context |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | paused | Public ❗️ |   |NO❗️ |
| └ | _pause | Internal 🔒 | 🛑  | whenNotPaused onlyPauser |
| └ | _unpause | Internal 🔒 | 🛑  | whenPaused onlyPauser |
||||||
| **AbstractUSDm** | Implementation | IERC20 |||
| └ | _approve | Internal 🔒 | 🛑  | |
| └ | _transfer | Internal 🔒 | 🛑  | |
||||||
| **Blacklistable** | Implementation | Ownable |||
| └ | isBlacklisted | External ❗️ |   |NO❗️ |
| └ | blacklist | External ❗️ | 🛑  | onlyBlacklister |
| └ | unBlacklist | External ❗️ | 🛑  | onlyBlacklister |
| └ | updateBlacklister | External ❗️ | 🛑  | onlyOwner |
||||||
| **USDm** | Implementation | AbstractUSDm, Ownable, Pausable,
Blacklistable, Rescuable |||
| └ | initialize | Public ❗ | 🛑  |NO❗ |
| └ | mint | External ❗ | 🛑  | whenNotPaused onlyMinters notBlacklisted
notBlacklisted |
| └ | minterAllowance | External ❗ |   |NO❗ |
| └ | isMinter | External ❗ |   |NO❗ |
| └ | allowance | External ❗ |   |NO❗ |
| └ | totalSupply | External ❗ |   |NO❗ |
| └ | balanceOf | External ❗ |   |NO❗ |
| └ | approve | External ❗ | 🛑  | whenNotPaused notBlacklisted
notBlacklisted |
| └ | _approve | Internal 🔒 | 🛑  | |
| └ | transferFrom | External ❗ | 🛑  | whenNotPaused notBlacklisted
notBlacklisted notBlacklisted |
| └ | transfer | External ❗ | 🛑  | whenNotPaused notBlacklisted
notBlacklisted |
| └ | _transfer | Internal 🔒 | 🛑  | |
| └ | configureMinter | External ❗ | 🛑  | whenNotPaused onlyMasterMinter |
| └ | removeMinter | External ❗ | 🛑  | onlyMasterMinter |
| └ | burn | External ❗ | 🛑  | whenNotPaused onlyMinters notBlacklisted |
| └ | updateMasterMinter | External ❗ | 🛑  | onlyOwner |
||||||
```

Legend

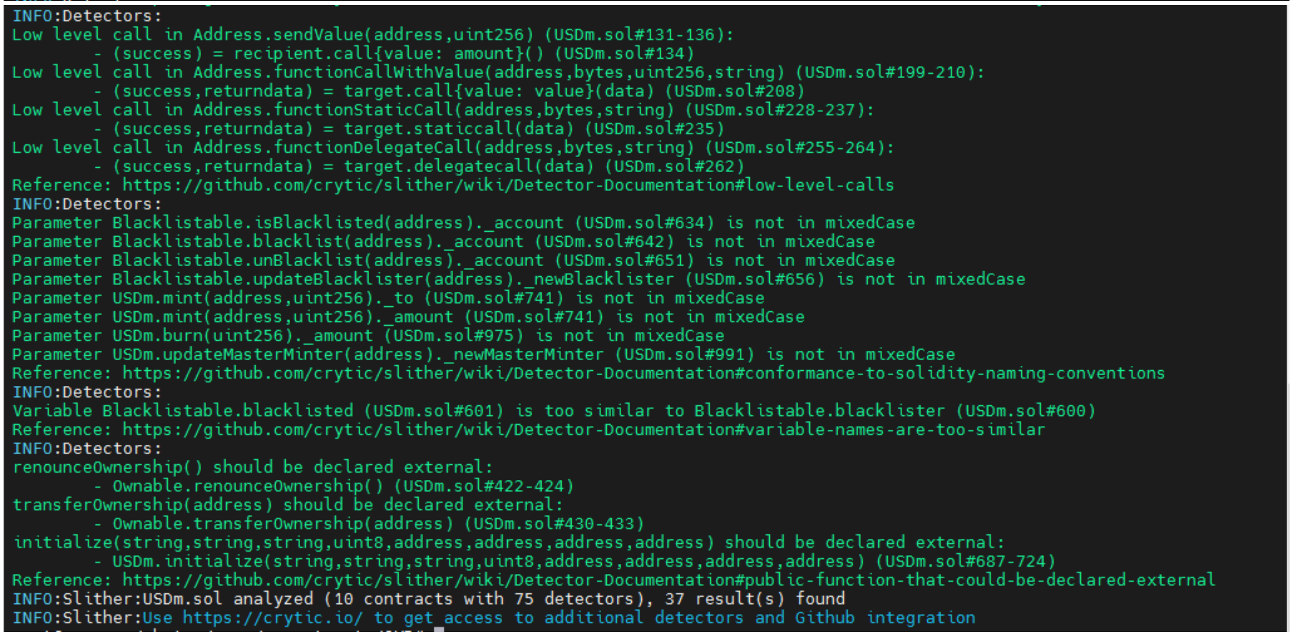
```
|  Symbol  |  Meaning  |
|:--------:|-----------|
|    🛑    | Function can modify state |
|    💵    | Function is payable |
```

Slither Results Log

Slither Log >> USDm.sol



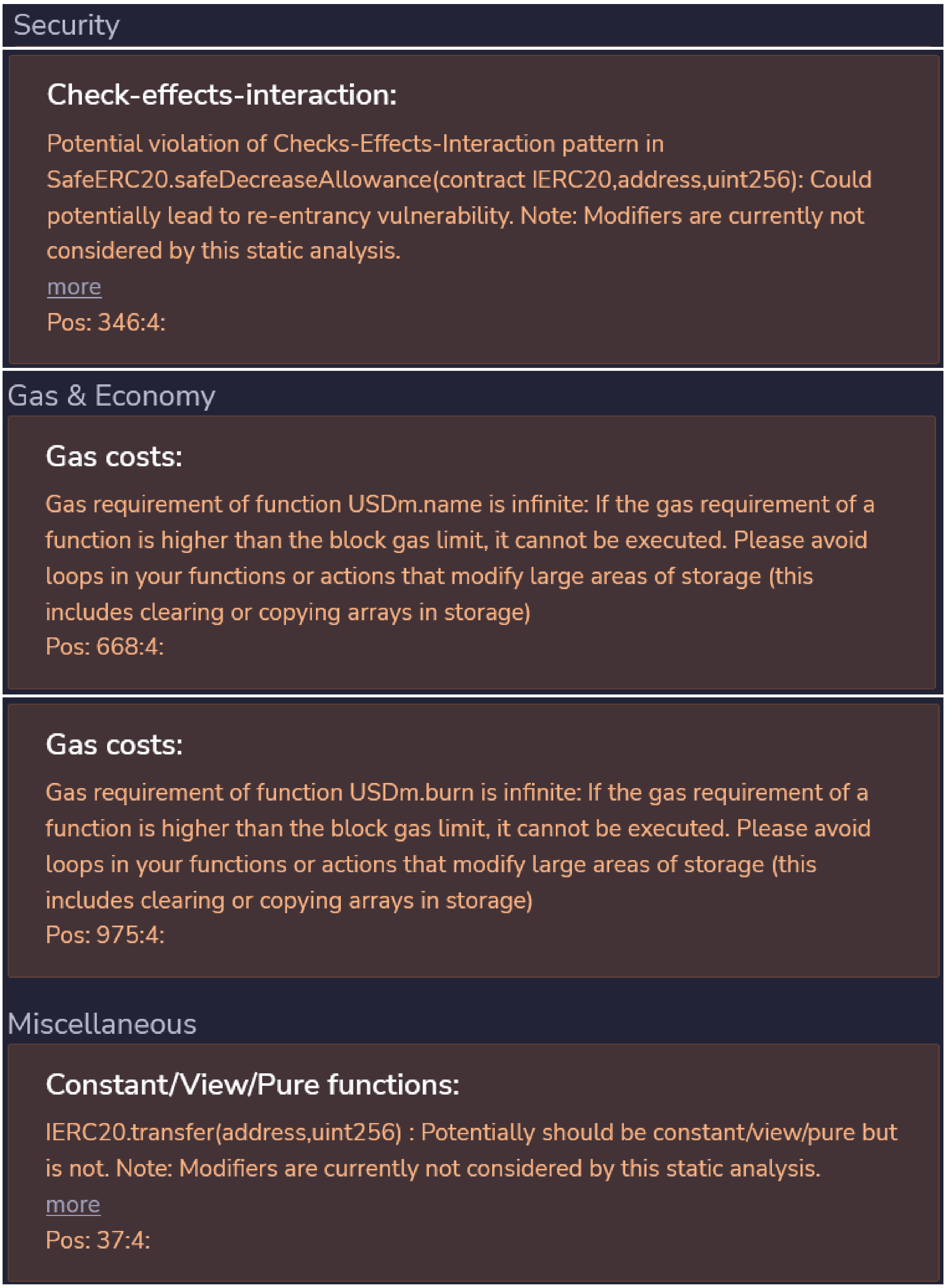
```
INFO:Detectors:
USDm.allowance(address,address).owner (USDm.sol#800) shadows:
        - Ownable.owner() (USDm.sol#403-405) (function)
USDm._approve(address,address,uint256).owner (USDm.sol#855) shadows:
        - Ownable.owner() (USDm.sol#403-405) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (USDm.sol#272-292) uses assembly
        - INLINE ASM (USDm.sol#284-287)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
AbstractUSDm._approve(address,address,uint256) (USDm.sol#586-590) is never used and should be removed
AbstractUSDm._transfer(address,address,uint256) (USDm.sol#592-596) is never used and should be removed
Address.functionCall(address,bytes) (USDm.sol#156-158) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (USDm.sol#185-191) is never used and should be removed
Address.functionDelegateCall(address,bytes) (USDm.sol#245-247) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (USDm.sol#255-264) is never used and should be removed
Address.functionStaticCall(address,bytes) (USDm.sol#218-220) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (USDm.sol#228-237) is never used and should be removed
Address.sendValue(address,uint256) (USDm.sol#131-136) is never used and should be removed
Context._msgData() (USDm.sol#383-385) is never used and should be removed
Pausable._pause() (USDm.sol#567-570) is never used and should be removed
Pausable._unpause() (USDm.sol#579-582) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (USDm.sol#322-335) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (USDm.sol#346-357) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (USDm.sol#337-344) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (USDm.sol#306-313) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (USDm.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```



```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (USDm.sol#131-136):
        - (success) = recipient.call{value: amount}() (USDm.sol#134)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (USDm.sol#199-210):
        - (success,returndata) = target.call{value: value}(data) (USDm.sol#208)
Low level call in Address.functionStaticCall(address,bytes,string) (USDm.sol#228-237):
        - (success,returndata) = target.staticcall(data) (USDm.sol#235)
Low level call in Address.functionDelegateCall(address,bytes,string) (USDm.sol#255-264):
        - (success,returndata) = target.delegatecall(data) (USDm.sol#262)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter Blacklistable.isBlacklisted(address)._account (USDm.sol#634) is not in mixedCase
Parameter Blacklistable.blacklist(address)._account (USDm.sol#642) is not in mixedCase
Parameter Blacklistable.unBlacklist(address)._account (USDm.sol#651) is not in mixedCase
Parameter Blacklistable.updateBlacklister(address)._newBlacklister (USDm.sol#656) is not in mixedCase
Parameter USDm.mint(address,uint256)._to (USDm.sol#741) is not in mixedCase
Parameter USDm.mint(address,uint256)._amount (USDm.sol#741) is not in mixedCase
Parameter USDm.burn(uint256)._amount (USDm.sol#975) is not in mixedCase
Parameter USDm.updateMasterMinter(address)._newMasterMinter (USDm.sol#991) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable Blacklistable.blacklisted (USDm.sol#601) is too similar to Blacklistable.blacklister (USDm.sol#600)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (USDm.sol#422-424)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (USDm.sol#430-433)
initialize(string,string,string,uint8,address,address,address,address) should be declared external:
        - USDm.initialize(string,string,string,uint8,address,address,address,address) (USDm.sol#687-724)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:USDm.sol analyzed (10 contracts with 75 detectors), 37 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

USDm.sol



Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SafeERC20.safeDecreaseAllowance(contract IERC20,address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 346:4:

Gas & Economy

Gas costs:

Gas requirement of function USDm.name 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: 668:4:

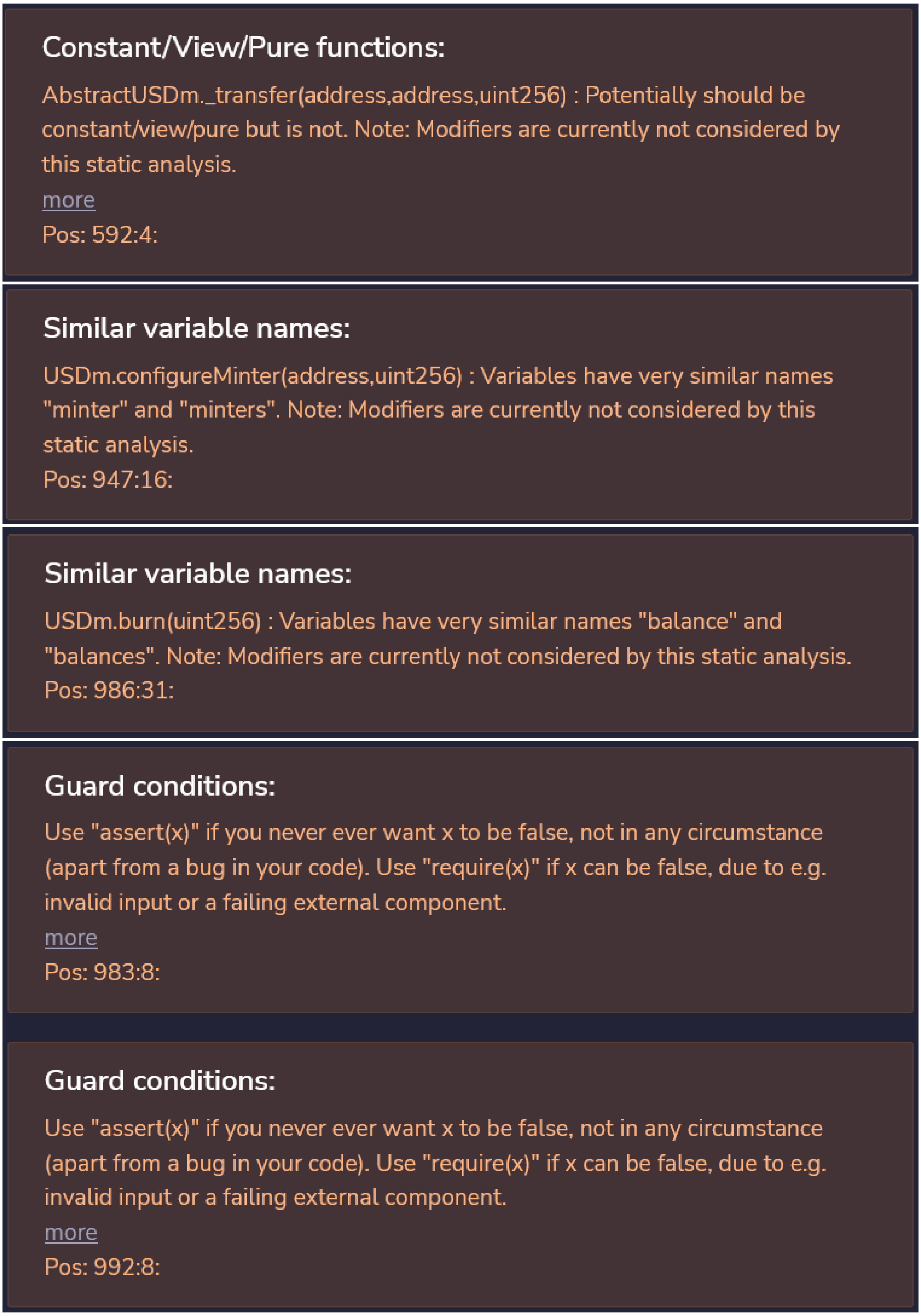
Gas costs:

Gas requirement of function USDm.burn 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: 975:4:

Miscellaneous

Constant/View/Pure functions:

IERC20.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 37:4:



Constant/View/Pure functions:

AbstractUSDm._transfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 592:4:

Similar variable names:

USDm.configureMinter(address,uint256) : Variables have very similar names "minter" and "minters". Note: Modifiers are currently not considered by this static analysis.
Pos: 947:16:

Similar variable names:

USDm.burn(uint256) : Variables have very similar names "balance" and "balances". Note: Modifiers are currently not considered by this static analysis.
Pos: 986:31:

Guard conditions:

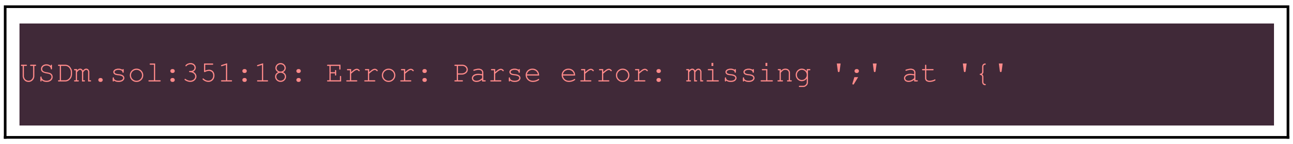
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 983:8:

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.
more
Pos: 992:8:

Solhint Linter

USDm.sol



```
USDm.sol:351:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

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

EtherAuthority