Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	First Digital USD
Website:	firstdigitallabs.com
Platform:	Ethereum
Language:	Solidity
Date:	May 9th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the First Digital USD Token smart contract from firstdigitom 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 May 9th, 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

- This Solidity smart contract is for a stablecoin, which is an ERC20 token with additional features such as permit, ownable functionality with a two-step upgradeable pattern, and pausable functionality. Here's a breakdown of its main features and functions:
- The contract inherits from ERC20PermitUpgradeable, Ownable2StepUpgradeable, and PausableUpgradeable.
- This contract provides the functionality described below:
    - The initialize function sets up the contract. It initializes the parent contracts and sets up the token name and symbol.
    - mint: Allows the owner to mint new tokens.
    - burn: Allows the owner to burn tokens.
    - freeze: Allows the owner to freeze an account.
    - unfreeze: Allows the owner to unfreeze an account.
    - pause: Allows the owner to pause token transfers.
    - unpause: Allows the owner to unpause token transfers.
- This contract provides functionality for creating and managing a stablecoin with features such as minting, burning, freezing accounts, pausing transfers, and permit functionality. It emphasizes security and control by allowing the owner to manage frozen accounts and control token transfers through pausing functionality.

Audit scope

Name	Code Review and Security Analysis Report for First Digital USD Token Smart Contract
Platform	Ethereum
Language	Solidity
File	Stablecoin. sol
Initial code link	0xc5f0f7b66764F6ec8C8Dff7BA683102295E16409
Audit Date	May 9th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: First Digital USD • Symbol: FDUSD • Decimals: 18	**YES, This is valid.**
Ownership control: • The owner can pause/unpause the contract state. • Burn amount. • Updates account to a frozen state. • Mint a new token. • The current owner can transfer the ownership. • The new owner accepts the ownership transfer.	**YES, This is valid.** **We suggest 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. A 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 0 low, and 1 very low level issues.

Investors' 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	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	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	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Yes
Pause Transfer?	Yes
Max Tax?	No
Is it Anti-whale?	No
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	Yes
Is it Proxy?	Yes
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in First Digital USD 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 First Digital USD Token.

The EtherAuthority team has no 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 First Digital USD Token smart contract code in the form of an Etherscan 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

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	write	initializer	No Issue
3	notFrozen	modifier	Passed	No Issue
4	mint	external	Centralization	Refer Audit Findings
5	burn	external	Centralization	Refer Audit Findings
6	freeze	external	Centralization	Refer Audit Findings
7	unfreeze	external	Centralization	Refer Audit Findings
8	pause	external	Centralization	Refer Audit Findings
9	unpause	external	Centralization	Refer Audit Findings
10	_transfer	internal	Passed	No Issue
11	_approve	internal	Passed	No Issue
12	__ERC20Permit_init	internal	access only Initializing	No Issue
13	__ERC20Permit_init_unchained	internal	access only Initializing	No Issue
14	permit	write	Passed	No Issue
15	nonces	read	Passed	No Issue
16	DOMAIN_SEPARATOR	external	Passed	No Issue
17	_useNonce	internal	Passed	No Issue
18	__EIP712_init	internal	access only Initializing	No Issue
19	__EIP712_init_unchained	internal	access only Initializing	No Issue
20	_domainSeparatorV4	internal	Passed	No Issue
21	_buildDomainSeparator	read	Passed	No Issue
22	_hashTypedDataV4	internal	Passed	No Issue
23	_EIP712NameHash	internal	Passed	No Issue
24	_EIP712VersionHash	internal	Passed	No Issue
25	__ERC20_init	internal	access only Initializing	No Issue
26	__ERC20_init_unchained	internal	access only Initializing	No Issue
27	name	read	Passed	No Issue
28	symbol	read	Passed	No Issue
29	decimals	read	Passed	No Issue
30	totalSupply	read	Passed	No Issue
31	balanceOf	read	Passed	No Issue
32	transfer	write	Passed	No Issue
33	allowance	read	Passed	No Issue
34	approve	write	Passed	No Issue
35	transferFrom	write	Passed	No Issue
36	increaseAllowance	write	Passed	No Issue

37	decreaseAllowance	write	Passed	No Issue
38	_transfer	internal	Passed	No Issue
39	_mint	internal	Passed	No Issue
40	_burn	internal	Passed	No Issue
41	_approve	internal	Passed	No Issue
42	_spendAllowance	internal	Passed	No Issue
43	_beforeTokenTransfer	internal	Passed	No Issue
44	_afterTokenTransfer	internal	Passed	No Issue
45	__Pausable_init	internal	access only Initializing	No Issue
46	__Pausable_init_unchained	internal	access only Initializing	No Issue
47	whenNotPaused	modifier	Passed	No Issue
48	whenPaused	modifier	Passed	No Issue
49	paused	read	Passed	No Issue
50	_requireNotPaused	internal	Passed	No Issue
51	_requirePaused	internal	Passed	No Issue
52	_pause	internal	Passed	No Issue
53	_unpause	internal	Passed	No Issue
54	__Ownable2Step_init	internal	access only Initializing	No Issue
55	__Ownable2Step_init_unchained	internal	access only Initializing	No Issue
56	pendingOwner	read	Passed	No Issue
57	transferOwnership	write	access only Owner	No Issue
58	_transferOwnership	internal	Passed	No Issue
59	acceptOwnership	external	Passed	No Issue
60	__Ownable_init	internal	access only Initializing	No Issue
61	__Ownable_init_unchained	internal	access only Initializing	No Issue
62	onlyOwner	modifier	Passed	No Issue
63	owner	read	Passed	No Issue
64	_checkOwner	internal	Passed	No Issue
65	renounceOwnership	write	access only Owner	No Issue
66	transferOwnership	write	access only Owner	No Issue
67	_transferOwnership	internal	Passed	No Issue
68	__Context_init	internal	access only Initializing	No Issue
69	__Context_init_unchained	internal	access only Initializing	No Issue
70	_msgSender	internal	Passed	No Issue
71	_msgData	internal	Passed	No Issue
72	initializer	modifier	Passed	No Issue
73	reinitializer	modifier	Passed	No Issue
74	onlyInitializing	modifier	Passed	No Issue
75	_disableInitializers	internal	Passed	No Issue
76	_getInitializedVersion	internal	Passed	No Issue
77	_isInitializing	internal	Passed	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 a 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 a 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.

Email: audit@EtherAuthority.io

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

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Centralization:

In the contract ifAdmin is an owner authority on the following function:

TransparentUpgradeableProxy.sol

- admin
- implementation
- changeAdmin
- upgradeTo
- upgradeToAndCall

Stablecoin.sol

- mint
- burn
- freeze
- unfreeze
- pause
- unpause

Resolution: We suggest carefully managing the ifAdmin private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

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 Admin functions:

Stablecoin.sol

- pause: The owner can trigger a stop.
- unpause: The owner can return to a normal state.
- mint: Mint a new amount by the owner.
- burn: Burn amount by the owner.
- freeze: Adds account to the frozen state by the owner.
- unfreeze: Removes account from frozen state by the owner.

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

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 Informational issue 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 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, and 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).

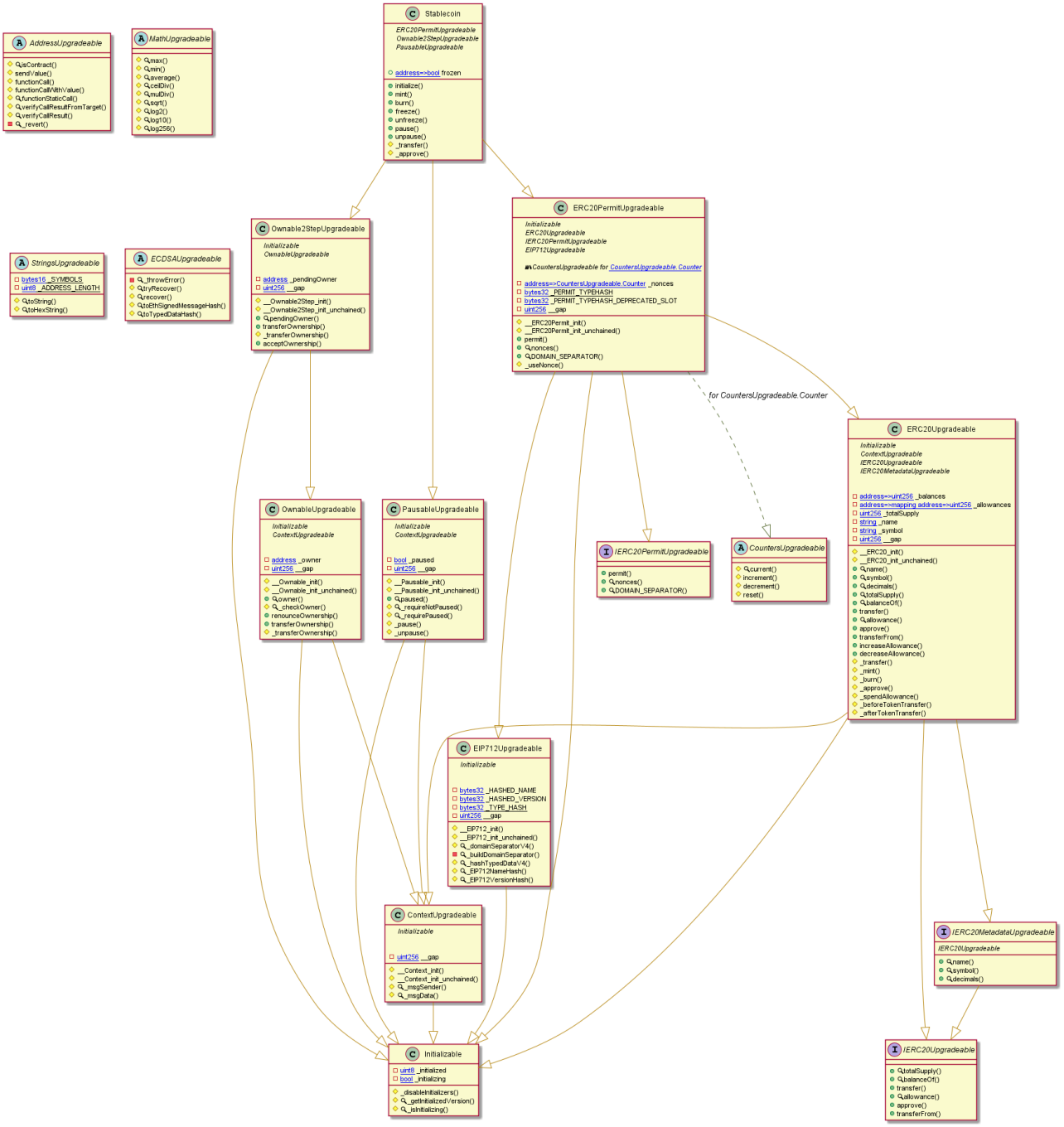
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 - First Digital USD Token



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.

Slither Log >> Stablecoin.sol

```
ERC20PermitUpgradeable.__ERC20Permit_init(string).name (Stablecoin.sol#997) shadows:
        - ERC20Upgradeable.name() (Stablecoin.sol#785-787) (function)
        - IERC20MetadataUpgradeable.name() (Stablecoin.sol#406) (function)
Stablecoin.initialize(string,string)._name (Stablecoin.sol#1051) shadows:
        - ERC20Upgradeable._name (Stablecoin.sol#773) (state variable)
Stablecoin.initialize(string,string)._symbol (Stablecoin.sol#1051) shadows:
        - ERC20Upgradeable._symbol (Stablecoin.sol#774) (state variable)
Stablecoin._approve(address,address,uint256).owner (Stablecoin.sol#1098) shadows:
        - OwnableUpgradeable.owner() (Stablecoin.sol#656-658) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Ownable2StepUpgradeable.transferOwnership(address).newOwner (Stablecoin.sol#697) lacks a zero-check on :
                - _pendingOwner = newOwner (Stablecoin.sol#698)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (Stablecoin.sol#1003-1022) uses timestamp f
or comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (Stablecoin.sol#1012)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable._revert(bytes,string) (Stablecoin.sol#105-114) uses assembly
        - INLINE ASM (Stablecoin.sol#107-110)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (Stablecoin.sol#169-222) uses assembly
        - INLINE ASM (Stablecoin.sol#177-181)
        - INLINE ASM (Stablecoin.sol#191-196)
        - INLINE ASM (Stablecoin.sol#200-206)
StringsUpgradeable.toString(uint256) (Stablecoin.sol#417-435) uses assembly
        - INLINE ASM (Stablecoin.sol#422-424)
        - INLINE ASM (Stablecoin.sol#427-429)
ECDSAUpgradeable.tryRecover(bytes32,bytes) (Stablecoin.sol#481-495) uses assembly
        - INLINE ASM (Stablecoin.sol#486-490)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
AddressUpgradeable._revert(bytes,string) (Stablecoin.sol#105-114) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (Stablecoin.sol#33-35) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (Stablecoin.sol#37-43) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (Stablecoin.sol#45-51) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (Stablecoin.sol#53-62) is never used and should be remove
d
AddressUpgradeable.functionStaticCall(address,bytes) (Stablecoin.sol#64-66) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (Stablecoin.sol#68-75) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (Stablecoin.sol#26-31) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (Stablecoin.sol#93-103) is never used and should be removed
AddressUpgradeable.verifyCallResultFromTarget(address,bool,bytes,string) (Stablecoin.sol#77-91) is never used and should be remo
ved
ContextUpgradeable.__Context_init_unchained() (Stablecoin.sol#625-626) is never used and should be removed
ContextUpgradeable._msgData() (Stablecoin.sol#631-633) is never used and should be removed
CountersUpgradeable.decrement(CountersUpgradeable.Counter) (Stablecoin.sol#133-139) is never used and should be removed
CountersUpgradeable.reset(CountersUpgradeable.Counter) (Stablecoin.sol#141-143) is never used and should be removed
ECDSAUpgradeable.recover(bytes32,bytes) (Stablecoin.sol#497-501) is never used and should be removed
ECDSAUpgradeable.recover(bytes32,bytes32,bytes32) (Stablecoin.sol#513-521) is never used and should be removed
ECDSAUpgradeable.toEthSignedMessageHash(bytes) (Stablecoin.sol#556-558) is never used and should be removed
ECDSAUpgradeable.toEthSignedMessageHash(bytes32) (Stablecoin.sol#552-554) is never used and should be removed
ECDSAUpgradeable.tryRecover(bytes32,bytes) (Stablecoin.sol#481-495) is never used and should be removed
ECDSAUpgradeable.tryRecover(bytes32,bytes32,bytes32) (Stablecoin.sol#503-511) is never used and should be removed
EIP712Upgradeable.__EIP712_init(string,string) (Stablecoin.sol#950-952) is never used and should be removed
ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (Stablecoin.sol#1001) is never used and should be removed
Initializable._disableInitializers() (Stablecoin.sol#604-610) is never used and should be removed
Initializable._getInitializedVersion() (Stablecoin.sol#612-614) is never used and should be removed
Initializable._isInitializing() (Stablecoin.sol#616-618) is never used and should be removed
MathUpgradeable.average(uint256,uint256) (Stablecoin.sol#161-163) is never used and should be removed
MathUpgradeable.ceilDiv(uint256,uint256) (Stablecoin.sol#165-167) is never used and should be removed
MathUpgradeable.log10(uint256) (Stablecoin.sol#308-340) is never used and should be removed
MathUpgradeable.log10(uint256,MathUpgradeable.Rounding) (Stablecoin.sol#342-347) is never used and should be removed
MathUpgradeable.log2(uint256) (Stablecoin.sol#263-299) is never used and should be removed
MathUpgradeable.log2(uint256,MathUpgradeable.Rounding) (Stablecoin.sol#301-306) is never used and should be removed
MathUpgradeable.log256(uint256) (Stablecoin.sol#349-373) is never used and should be removed
MathUpgradeable.log256(uint256,MathUpgradeable.Rounding) (Stablecoin.sol#375-380) is never used and should be removed
MathUpgradeable.max(uint256,uint256) (Stablecoin.sol#153-155) is never used and should be removed
```

```
MathUpgradeable.min(uint256,uint256) (Stablecoin.sol#157-159) is never used and should be removed
MathUpgradeable.mulDiv(uint256,uint256,uint256) (Stablecoin.sol#169-222) is never used and should be removed
MathUpgradeable.mulDiv(uint256,uint256,uint256,MathUpgradeable.Rounding) (Stablecoin.sol#224-235) is never used and should be re
moved
MathUpgradeable.sqrt(uint256) (Stablecoin.sol#237-254) is never used and should be removed
MathUpgradeable.sqrt(uint256,MathUpgradeable.Rounding) (Stablecoin.sol#256-261) is never used and should be removed
Ownable2StepUpgradeable.__Ownable2Step_init_unchained() (Stablecoin.sol#687-688) is never used and should be removed
OwnableUpgradeable.__Ownable_init() (Stablecoin.sol#643-645) is never used and should be removed
StringsUpgradeable.toHexString(address) (Stablecoin.sol#455-457) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (Stablecoin.sol#437-441) is never used and should be removed
StringsUpgradeable.toHexString(uint256,uint256) (Stablecoin.sol#443-453) is never used and should be removed
StringsUpgradeable.toString(uint256) (Stablecoin.sol#417-435) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (Stablecoin.sol#2) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (Stablecoin.sol#26-31):
        - (success) = recipient.call{value: amount}() (Stablecoin.sol#29)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (Stablecoin.sol#53-62):
        - (success,returndata) = target.call{value: value}(data) (Stablecoin.sol#60)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (Stablecoin.sol#68-75):
        - (success,returndata) = target.staticcall(data) (Stablecoin.sol#73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (Stablecoin.sol#17) is not in mixedCase
Function ContextUpgradeable.__Context_init() (Stablecoin.sol#622-623) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (Stablecoin.sol#625-626) is not in mixedCase
Variable ContextUpgradeable.__gap (Stablecoin.sol#635) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (Stablecoin.sol#643-645) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (Stablecoin.sol#647-649) is not in mixedCase
Variable OwnableUpgradeable.__gap (Stablecoin.sol#679) is not in mixedCase
Function Ownable2StepUpgradeable.__Ownable2Step_init() (Stablecoin.sol#683-685) is not in mixedCase
Function Ownable2StepUpgradeable.__Ownable2Step_init_unchained() (Stablecoin.sol#687-688) is not in mixedCase
Variable Ownable2StepUpgradeable.__gap (Stablecoin.sol#713) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (Stablecoin.sol#723-725) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (Stablecoin.sol#727-729) is not in mixedCase
Variable PausableUpgradeable.__gap (Stablecoin.sol#763) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (Stablecoin.sol#776-778) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (Stablecoin.sol#780-783) is not in mixedCase
Variable ERC20Upgradeable.__gap (Stablecoin.sol#940) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init(string,string) (Stablecoin.sol#950-952) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init_unchained(string,string) (Stablecoin.sol#954-959) is not in mixedCase
Function EIP712Upgradeable._EIP712NameHash() (Stablecoin.sol#977-979) is not in mixedCase
Function EIP712Upgradeable._EIP712VersionHash() (Stablecoin.sol#981-983) is not in mixedCase
Variable EIP712Upgradeable._HASHED_NAME (Stablecoin.sol#945) is not in mixedCase
Variable EIP712Upgradeable._HASHED_VERSION (Stablecoin.sol#946) is not in mixedCase
Variable EIP712Upgradeable.__gap (Stablecoin.sol#985) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init(string) (Stablecoin.sol#997-999) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (Stablecoin.sol#1001) is not in mixedCase
Function ERC20PermitUpgradeable.DOMAIN_SEPARATOR() (Stablecoin.sol#1028-1030) is not in mixedCase
Variable ERC20PermitUpgradeable._PERMIT_TYPEHASH_DEPRECATED_SLOT (Stablecoin.sol#995) is not in mixedCase
Variable ERC20PermitUpgradeable.__gap (Stablecoin.sol#1038) is not in mixedCase
Parameter Stablecoin.initialize(string,string)._name (Stablecoin.sol#1051) is not in mixedCase
Parameter Stablecoin.initialize(string,string)._symbol (Stablecoin.sol#1051) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Stablecoin.sol analyzed (17 contracts with 84 detectors), 98 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.

Stablecoin.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 173:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 838:16:

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.

more

Pos: 1854:23:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

Pos: 180:50:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function Stablecoin.unpause 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: 1980:11:

Constant/View/Pure functions:

Stablecoin._approve(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 2000:11:

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: 1922:15:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 1964:15:

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.

Stablecoin.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Error message for require is too long
Pos: 9:108
Error message for require is too long
Pos: 9:178
Error message for require is too long
Pos: 9:1018
Error message for require is too long
Pos: 9:1052
Error message for require is too long
Pos: 9:1065
Error message for require is too long
Pos: 9:1078
Function name must be in mixedCase
Pos: 5:1101
Code contains empty blocks
Pos: 57:1101
Function name must be in mixedCase
Pos: 5:1104
Code contains empty blocks
Pos: 67:1104
Function name must be in mixedCase
Pos: 5:1130
Function name must be in mixedCase
Pos: 5:1134
Error message for require is too long
Pos: 9:1176
Function name must be in mixedCase
Pos: 5:1199
Function name must be in mixedCase
Pos: 5:1203
Code contains empty blocks
Pos: 72:1203
Error message for require is too long
Pos: 9:1239
Function name must be in mixedCase
Pos: 5:1267
Function name must be in mixedCase
Pos: 5:1271
Function name must be in mixedCase
Pos: 5:1371
Function name must be in mixedCase
```

```
Pos: 5:1375
Error message for require is too long
Pos: 9:1525
Error message for require is too long
Pos: 9:1552
Error message for require is too long
Pos: 9:1553
Error message for require is too long
Pos: 9:1558
Error message for require is too long
Pos: 9:1607
Error message for require is too long
Pos: 9:1612
Error message for require is too long
Pos: 9:1642
Error message for require is too long
Pos: 9:1643
Code contains empty blocks
Pos: 24:1689
Code contains empty blocks
Pos: 24:1709
Function name must be in mixedCase
Pos: 5:1740
Function name must be in mixedCase
Pos: 5:1744
Function name must be in mixedCase
Pos: 5:1791
Function name must be in mixedCase
Pos: 5:1801
Function name must be in mixedCase
Pos: 5:1835
Function name must be in mixedCase
Pos: 5:1839
Code contains empty blocks
Pos: 84:1839
Avoid making time-based decisions in your business logic
Pos: 17:1853
```

Software analysis result:

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

Email: audit@EtherAuthority.io