Ether Authority
etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: USDC Token
Website: circle.com/en/usdc
Platform: Base Chain Network
Language: Solidity
Date: May 29th, 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 USDC token smart contract from circle.com 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 29th, 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

USD Coin (USDC) is a digital dollar stablecoin issued by Circle, fully backed by US dollar reserves. It offers global, near-instant, low-cost transactions and is regulated with transparent monthly reserve attestations. USDC is widely used across numerous blockchain networks and is designed to maintain a 1:1 value with the US dollar, making it a stable and secure digital currency. Businesses and individuals can easily mint and redeem USDC, ensuring liquidity and reliability.

Code Details

- This Solidity code defines a token contract called FiatTokenV2_2, which is an upgraded version of the original FiatToken contract. Let's break down its key features:
    - **Versioning:** The contract has multiple versions (FiatTokenV1, FiatTokenV1_1, FiatTokenV2, FiatTokenV2_1, FiatTokenV2_2). Each version introduces new functionalities or upgrades existing ones. The versioning allows for smooth upgrades without disrupting existing functionality.
    - **Initial Setup:** The contract initializes various parameters such as name, symbol, currency, decimals, master minter, pauser, blacklister, and owner. It ensures that these parameters are set correctly during contract deployment.
    - **Minting and Burning:** The contract supports minting and burning of tokens. Minters are designated addresses that can mint new tokens, subject to an allowance specified by the master minter.
    - **Token Transfers:** It facilitates token transfers between addresses. The transfer, transferFrom, and approve functions handle standard ERC-20 token transfer functionality.
    - **Authorization:** The contract implements ERC-3009 and ERC-2612 standards for token authorization. Users can approve spending tokens on their behalf without interacting with the contract directly. This functionality enhances security and usability.
    - **Blacklisting:** The contract allows for blacklisting specific accounts, preventing them from sending or receiving tokens. This feature is useful for compliance and security purposes.
    - **Pausing:** The contract can be paused and unpaused by the pauser address. When paused, token transfers are disabled, adding an extra layer of security and control.
    - **Rescue Functionality:** The contract includes a rescuer address that can recover ERC-20 tokens mistakenly sent to the contract address. This feature prevents tokens from being lost irreversibly.
    - **Domain Separation:** The contract uses domain separation for enhanced security in signature verification. It generates a unique domain separator for each version of the contract.

- **Upgradeability:** The contract design allows for future upgrades by introducing new versions (FiatTokenV2_1, FiatTokenV2_2). Upgrades can introduce new functionalities or fix bugs while maintaining compatibility with existing applications.

- Overall, the FiatTokenV2_2 contract provides a comprehensive set of features for managing and transferring ERC-20 tokens, with a focus on security, flexibility, and upgradability.

Audit scope

Name	**Code Review and Security Analysis Report for USDC Token Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	FiatTokenV2_2.sol
Smart Contract Code	0x2ce6311ddae708829bc0784c967b7d77d19fd779
Audit Date	May 29th,2024
Audit Result	**Passed**

Code Audit History

2	0	0	0	0	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
0	**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.

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Master Minter Specifications: • Adds or updates a new minter address with a mint allowance. • Removes a minter address. **Rescuer Specifications:** • Rescue ERC20 tokens are locked up in this contract. **Blacklister Specifications:** • Add/Remove account to blacklist addresses. **Pausable Specifications:** • The owner calls to pause, which triggers the stopped state. • The owner calls to unpause and returns to a normal state.	**YES, This is valid.**
Owner Specifications: • Update the master minter address. • Update the rescuer address. • Update the blacklister address. • Updates the pauser address. • 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, 0 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	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	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Yes
Blacklist Check	Yes
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**

Email: audit@EtherAuthority.io

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 USDC 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 USDC Token.

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 USDC Token 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

FiatTokenV2_2.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializeV2_2	write	Missing required error message	Refer Audit Findings
3	_chainId2	internal	Passed	No Issue
4	_domainSeparator	internal	Passed	No Issue
5	permit	external	Passed	No Issue
6	transferWithAuthorization	external	Passed	No Issue
7	receiveWithAuthorization	external	Passed	No Issue
8	cancelAuthorization	external	Passed	No Issue
9	_setBlacklistState	external	Passed	No Issue
10	_setBalance	internal	Passed	No Issue
11	_isBlacklisted	internal	Passed	No Issue
12	_balanceOf	internal	Passed	No Issue
13	approve	external	Passed	No Issue
14	permit	external	Passed	No Issue
15	increaseAllowance	external	Passed	No Issue
16	decreaseAllowance	external	Passed	No Issue
17	initializeV2_1	external	Passed	No Issue
18	version	external	Passed	No Issue
19	initializeV2	external	Passed	No Issue
20	increaseAllowance	external	Passed	No Issue
21	decreaseAllowance	external	Passed	No Issue
22	transferWithAuthorization	external	Passed	No Issue
23	receiveWithAuthorization	external	Passed	No Issue
24	cancelAuthorization	external	Passed	No Issue
25	permit	external	Passed	No Issue
26	_increaseAllowance	internal	Passed	No Issue
27	_decreaseAllowance	internal	Passed	No Issue
28	initialize	write	Passed	No Issue
29	onlyMinters	modifier	Passed	No Issue
30	mint	external	access only Minters	No Issue
31	onlyMasterMinter	modifier	Passed	No Issue
32	minterAllowance	external	Passed	No Issue
33	isMinter	external	Passed	No Issue
34	allowance	external	Passed	No Issue
35	totalSupply	external	Passed	No Issue
36	balanceOf	external	Passed	No Issue
37	approve	external	Passed	No Issue
38	_approve	internal	Passed	No Issue

39	transferFrom	external	Passed	No Issue
40	transfer	external	Passed	No Issue
41	_transfer	internal	Passed	No Issue
42	configureMinter	external	access only Master Minter	No Issue
43	removeMinter	external	access only Master Minter	No Issue
44	burn	external	access only Minters	No Issue
45	updateMasterMinter	external	Centralized Ownership and Privileges Management	Refer Audit Findings
46	_blacklist	internal	Passed	No Issue
47	_unBlacklist	internal	Passed	No Issue
48	_setBlacklistState	internal	Passed	No Issue
49	_setBalance	internal	Passed	No Issue
50	_isBlacklisted	internal	Passed	No Issue
51	_balanceOf	internal	Passed	No Issue
52	nonces	external	Passed	No Issue
53	_permit	internal	Passed	No Issue
54	_permit	internal	Passed	No Issue
55	authorizationState	external	Passed	No Issue
56	_transferWithAuthorization	internal	Passed	No Issue
57	_transferWithAuthorization	internal	Passed	No Issue
58	_receiveWithAuthorization	internal	Passed	No Issue
59	_receiveWithAuthorization	internal	Passed	No Issue
60	_cancelAuthorization	internal	Passed	No Issue
61	_cancelAuthorization	internal	Passed	No Issue
62	_requireValidSignature	read	Passed	No Issue
63	_requireUnusedAuthorization	internal	Passed	No Issue
64	_requireValidAuthorization	read	Passed	No Issue
65	_markAuthorizationAsUsed	write	Passed	No Issue
66	_increaseAllowance	internal	Passed	No Issue
67	_decreaseAllowance	internal	Passed	No Issue
68	rescuer	external	Passed	No Issue
69	onlyRescuer	modifier	Passed	No Issue
70	rescueERC20	external	access only Rescuer	No Issue
71	updateRescuer	external	Centralized Ownership and Privileges Management	Refer Audit Findings
72	onlyBlacklister	modifier	Passed	No Issue
73	notBlacklisted	modifier	Passed	No Issue
74	isBlacklisted	external	Passed	No Issue
75	blacklist	external	Centralized Ownership	Refer Audit Findings

			and Privileges Management	
76	unBlacklist	external	Centralized Ownership and Privileges Management	Refer Audit Findings
77	updateBlacklister	external	Centralized Ownership and Privileges Management	Refer Audit Findings
78	_isBlacklisted	internal	Passed	No Issue
79	_blacklist	internal	Passed	No Issue
80	_unBlacklist	internal	Passed	No Issue
81	whenNotPaused	modifier	Passed	No Issue
82	onlyPauser	modifier	Passed	No Issue
83	pause	external	access only Pauser	No Issue
84	unpause	external	access only Pauser	No Issue
85	updatePauser	external	Centralized Ownership and Privileges Management	Refer Audit Findings
86	owner	external	Passed	No Issue
87	setOwner	internal	Passed	No Issue
88	onlyOwner	modifier	Passed	No Issue
89	transferOwnership	external	Centralized Ownership and Privileges Management	Refer Audit Findings
90	DOMAIN_SEPARATOR	external	Passed	No Issue
91	_domainSeparator	internal	Passed	No Issue
92	_approve	internal	Passed	No Issue
93	_transfer	internal	Passed	No Issue

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:

[I-01] Centralization:

Description:

In this contract Blacklistable.sol "blacklister" has authority to the following function:

- blacklist
- unBlacklist

Blacklister can blacklist any user.

This contract onlyOwner has authority to the following function:

- updateBlacklister
- updateMasterMinter
- transferOwnership
- updatePauser
- updateRescuer

Recommendation: We suggest carefully managing the onlyOwner account's private keys 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.
e.g., Multisignature wallets.

[I-02] Missing required error message:

```
contract FiatTokenV2_2 is FiatTokenV2_1 {
    /**
     * @notice Initialize v2.2
     * @param accountsToBlacklist   A list of accounts to migrate from the old blacklist
     * @param newSymbol             New token symbol
     * data structure to the new blacklist data structure.
     */
    function initializeV2_2(
        address[] calldata accountsToBlacklist,
        string calldata newSymbol
    ) external {
        // solhint-disable-next-line reason-string
        require(_initializedVersion == 2);

        // Update fiat token symbol
        symbol = newSymbol;
```

Description:
There is no error message required.
Recommendation: We suggest setting relevant error messages to identify the failure of the transaction.

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.

You are here 

The following are Admin functions:

FiatTokenV1.sol

- configureMinter: Adds or updates a new minter address with a mint allowance only by the Master Minter of the owner.
- removeMinter: Removes a minter address only by the Master Minter of the owner.
- updateMasterMinter: The owner can update the master minter address.

Rescuable.sol

- rescueERC20: Rescue ERC20 tokens are locked up in this contract only by the Rescuer of the owner.
- updateRescuer: The owner can update the rescuer address.

Blacklistable.sol

- blacklist: Adds account to blacklist only by the Blacklister of the owner.
- unBlacklist: Removes account from blacklist only by the Blacklister of the owner.
- updateBlacklister: The owner can update the blacklister address.

Pausable.sol

- pause: The owner calls to pause, which triggers the stopped state.
- unpause: The owner calls to unpause, and returns to normal state.
- updatePauser: Updates the pauser address only 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 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.

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".**

Email: audit@EtherAuthority.io

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).

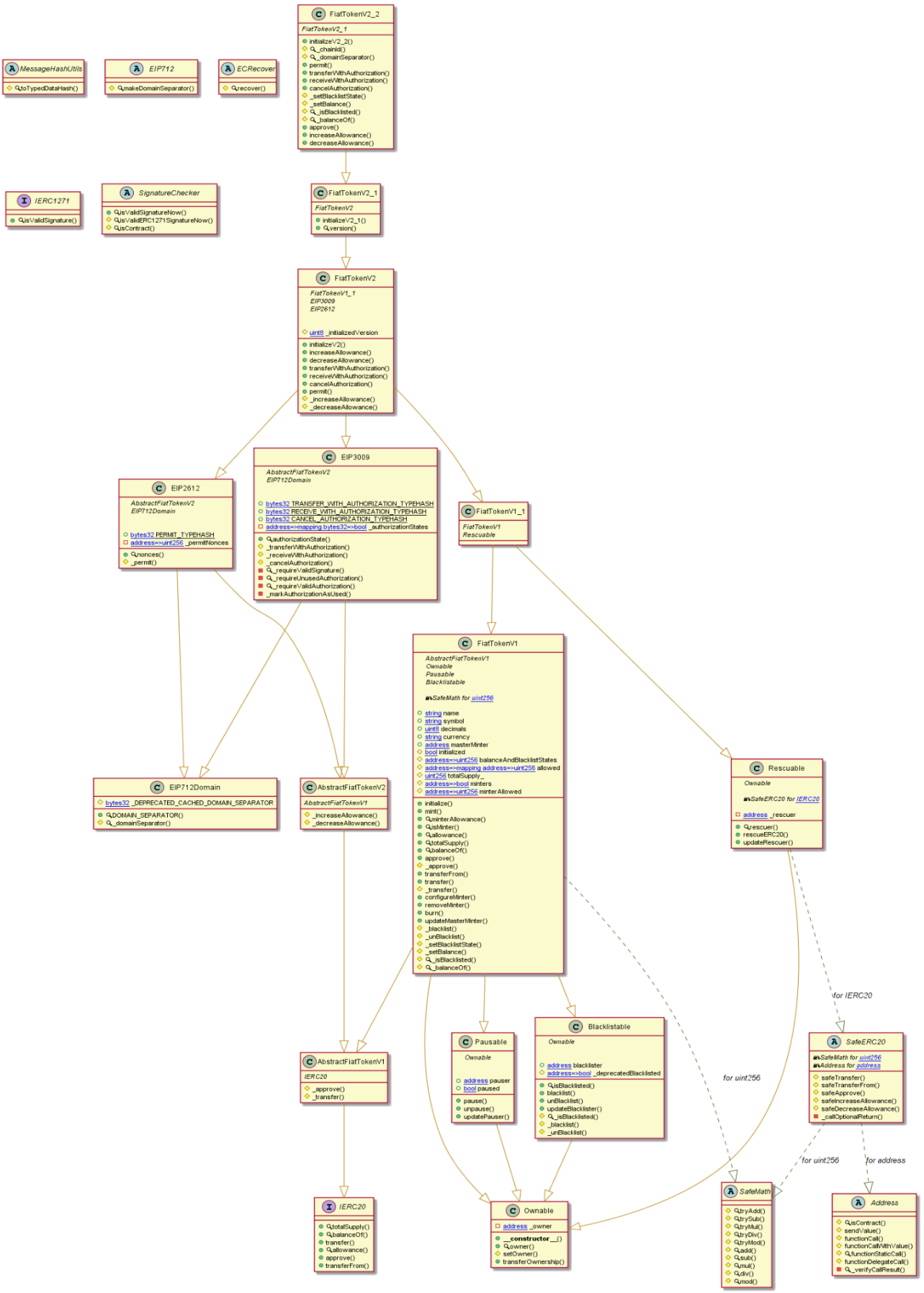
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

Code Flow Diagram - USDC 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.

FiatTokenV2_2.sol

```
INFO:Detectors:
FiatTokenV2_2.permit(address,address,uint256,uint256,bytes).owner (FiatTokenV2_2.sol#2232) shadows:
	- Ownable.owner() (FiatTokenV2_2.sol#812-814) (function)
FiatTokenV2_2.permit(address,address,uint256,uint256,uint8,bytes32,bytes32).owner (FiatTokenV2_2.sol#2409) shadows:
	- Ownable.owner() (FiatTokenV2_2.sol#812-814) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
EIP3009._requireValidAuthorization(address,bytes32,uint256,uint256) (FiatTokenV2_2.sol#1367-1379) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now > validAfter,FiatTokenV2: authorization is not yet valid) (FiatTokenV2_2.sol#1373-1376)
	- require(bool,string)(now < validBefore,FiatTokenV2: authorization is expired) (FiatTokenV2_2.sol#1377)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
FiatTokenV2_2._chainId() (FiatTokenV2_2.sol#2207-2213) uses assembly
	- INLINE ASM (FiatTokenV2_2.sol#2209-2211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
FiatTokenV2_2.initializeV2_2(address[],string) (FiatTokenV2_2.sol#2177-2201) has costly operations inside a loop:
	- delete _deprecatedBlacklisted[accountsToBlacklist[i]] (FiatTokenV2_2.sol#2195)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
FiatTokenV1._balanceOf(address) (FiatTokenV2_2.sol#1899-1906) is never used and should be removed
```

```
FiatTokenV1._isBlacklisted(address) (FiatTokenV2_2.sol#1884-1892) is never used and should be removed
FiatTokenV1._setBalance(address,uint256) (FiatTokenV2_2.sol#1877-1879) is never used and should be removed
FiatTokenV1._setBlacklistState(address,bool) (FiatTokenV2_2.sol#1865-1870) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (FiatTokenV2_2.sol#4) is too complex
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in SignatureChecker.isValidERC1271SignatureNow(address,bytes32,bytes) (FiatTokenV2_2.sol#712-728):
        - (success,result) = signer.staticcall(abi.encodeWithSelector(IERC1271.isValidSignature.selector,digest,signature)) (FiatTokenV2_2.sol#717-723)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function EIP712Domain.DOMAIN_SEPARATOR() (FiatTokenV2_2.sol#767-769) is not in mixedCase
Parameter FiatTokenV1.mint(address,uint256)._amount (FiatTokenV2_2.sol#1578) is not in mixedCase
Parameter FiatTokenV1.burn(uint256)._amount (FiatTokenV2_2.sol#1817) is not in mixedCase
Parameter FiatTokenV1.updateMasterMinter(address)._newMasterMinter (FiatTokenV2_2.sol#1837) is not in mixedCase
Contract FiatTokenV1_1 (FiatTokenV2_2.sol#1914-1916) is not in CapWords
Contract FiatTokenV2_1 (FiatTokenV2_2.sol#2139-2164) is not in CapWords
Function FiatTokenV2_1.initializeV2_1(address) (FiatTokenV2_2.sol#2144-2155) is not in mixedCase
Contract FiatTokenV2_2 (FiatTokenV2_2.sol#2170-2446) is not in CapWords
Function FiatTokenV2_2.initializeV2_2(address[],string) (FiatTokenV2_2.sol#2177-2201) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:FiatTokenV2_2.sol analyzed (23 contracts with 93 detectors), 66 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.

FiatTokenV2_2.sol

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.
Pos: 497:4:

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.
Pos: 2209:11:

Block timestamp:
Use of "now": "now" does not mean current time. "now" is an alias for "block.timestamp". "block.timestamp" can be influenced by miners to a certain degree, be careful.
Pos: 1454:60:

Low level calls:
Use of "delegatecall": should be avoided whenever possible. External code that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.
Pos: 438:50:

Gas costs:
Gas requirement of function FiatTokenV2_2.initializeV2_2 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: 2177:7:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

Pos: 2189:11:

Constant/View/Pure functions:FiatTokenV2_2._domainSeparator() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
Pos: 2218:7:

Similar variable names:FiatTokenV1.minterAllowance(address) : Variables have very similar names "minter" and "minters". Note: Modifiers are currently not considered by this static analysis.
Pos: 1620:32:

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: 2355:11:

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.
Pos: 2198:11:

Data truncated:Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 255:15:

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.

FiatTokenV2_2.sol

```
Compiler version >=0.6.0 <0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 58:1453
Error message for require is too long
Pos: 9:1823
Error message for require is too long
Pos: 9:1824
Error message for require is too long
Pos: 9:1837
Code contains empty blocks
Pos: 1:1913
Contract name must be in CamelCase
Pos: 1:1913
Contract name must be in CamelCase
Pos: 1:2138
Function name must be in mixedCase
Pos: 5:2143
Contract name must be in CamelCase
Pos: 1:2169
Function name must be in mixedCase
Pos: 5:2176
Error message for require is too long
Pos: 13:2189
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:2208
Error message for require is too long
Pos: 9:2350
Error message for require is too long
Pos: 9:2354
```

Software analysis result:

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

Email: audit@EtherAuthority.io