Ether Authority
etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	**USD+**
Website:	**overnight.fi**
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 USD+ smart contract from overnight.fi 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

Overnight.fi is a platform focused on optimizing returns for stablecoin assets. It offers users a way to earn a yield on their digital assets by leveraging various decentralized finance (DeFi) protocols. The website emphasizes security and ease of use, aiming to provide a stable and reliable income stream for users without the need to actively manage their investments.

Code Details

- The solidity smart contract code provided is an implementation of an upgradeable ERC20 token with special features such as rebasing and non-rebasing supply management. This contract, named `UsdPlusToken`, incorporates several OpenZeppelin libraries and utilizes a proxy pattern for upgrades.
- **Imports:** Essential OpenZeppelin libraries are imported for ERC20 token standard, access control, upgradeability, and utility functions.
- **Minting and Burning:**

    - Functions to mint and burn tokens, updating supply and balances.
    - Restricted to the exchanger role to ensure only authorized contracts can mint/burn tokens.
- **Rebasing Management:**
    - Functions to opt-in and opt-out of rebasing, which adjust the account's credits and the overall non-rebasing supply.
    - Internal `_isNonRebasingAccount` checks the rebasing state of an account.
- **Upgradeability:**
    - `_authorizeUpgrade` function to handle contract upgrades securely, restricted to the admin role.

Audit scope

Name	**Code Review and Security Analysis Report for USD+ Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	UsdPlusToken.sol
Smart Contract Code	0x8de5410692c0bc722695f17ca4dd55c9506052c6
Audit Date	May 29th,2024
Audit Result	**Passed**

Code Audit History

5	0	0	0	0	5
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
5	**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
Specifications: • Admin roles upgrade by default admin. • The exchanger address can be set by the admin. • The Payout Manager address can be set by the admin. • The Manager address can be set by the admin. • Pause/Unpause status set true by the Portfolio Agent. • Mints new tokens, increasing total supply only by the Exchanger. • Burns tokens, decreasing total Supply only by the only Exchanger. • Modify the supply without minting new tokens only by the Exchanger.	**YES, This is valid.**

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 5 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	Moderated
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?	Yes
Pause Transfer?	Yes
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy Contract?	Yes
Is it used Open Source?	Yes
External Call Risk?	No
Balance Modifiable?	No
Can Take Ownership?	Yes
Ownership Renounce?	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. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in USD+ 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 USD+.

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 USD+ 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

UsdPlusToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	nonReentrant	modifier	Passed	No Issue
3	initialize	write	Compile time warnings	Refer Audit Findings
4	_authorizeUpgrade	internal	DEFAULT_ADMIN_ROLE	No Issue
5	onlyExchanger	modifier	Passed	No Issue
6	onlyPayoutManager	modifier	Passed	No Issue
7	onlyPortfolioAgent	modifier	Passed	No Issue
8	onlyAdmin	modifier	Passed	No Issue
9	notPaused	modifier	Passed	No Issue
10	setExchanger	external	access only Admin	No Issue
11	setPayoutManager	external	access only Admin	No Issue
12	setRoleManager	external	access only Admin	No Issue
13	pause	write	access only Portfolio Agent	No Issue
14	unpause	write	access only Portfolio Agent	No Issue
15	name	write	Compile time warnings	Refer Audit Findings
16	symbol	write	Compile time warnings	Refer Audit Findings
17	decimals	read	Passed	No Issue
18	ownerLength	external	Passed	No Issue
19	nonRebaseOwnersLength	external	Passed	No Issue
20	ownerAt	external	Passed	No Issue
21	ownerBalanceAt	external	Passed	No Issue
22	totalSupplyOwners	external	Passed	No Issue
23	totalSupply	read	Passed	No Issue
24	rebasingCreditsPerToken	read	Passed	No Issue
25	rebasingCredits	read	Passed	No Issue
26	rebasingCreditsPerTokenHighres	read	Passed	No Issue
27	rebasingCreditsHighres	read	Passed	No Issue
28	balanceOf	read	Passed	No Issue
29	creditsBalanceOf	read	Passed	No Issue
30	creditsBalanceOfHighres	read	Passed	No Issue
31	transfer	write	Passed	No Issue

32	assetToCredit	read	Missing-zero-address-validation	Refer Audit Findings
33	creditToAsset	read	Missing-zero-address-validation	Refer Audit Findings
34	subCredits	read	Missing-zero-address-validation	Refer Audit Findings
35	transferFrom	write	Passed	No Issue
36	_executeTransfer	internal	Passed	No Issue
37	allowance	read	Passed	No Issue
38	approve	write	Passed	No Issue
39	increaseAllowance	write	Passed	No Issue
40	decreaseAllowance	write	Passed	No Issue
41	mint	external	access only exchanger	No Issue
42	_mint	internal	Passed	No Issue
43	burn	external	access only exchanger	No Issue
44	_burn	internal	Passed	No Issue
45	_creditsPerToken	internal	Passed	No Issue
46	_isNonRebasingAccount	internal	Compile time warnings	Refer Audit Findings
47	rebaseOptIn	write	access only Payout Manager	No Issue
48	rebaseOptOut	write	access only Payout Manager	No Issue
49	changeSupply	external	access only exchanger	No Issue
50	_beforeTokenTransfer	internal	Passed	No Issue
51	_afterTokenTransfer	internal	Passed	No Issue
52	__AccessControl_init	internal	access only Initializing	No Issue
53	__AccessControl_init_unchained	internal	access only Initializing	No Issue
54	onlyRole	modifier	Passed	No Issue
55	supportsInterface	read	Passed	No Issue
56	hasRole	read	Passed	No Issue
57	_checkRole	internal	Passed	No Issue
58	_checkRole	internal	Passed	No Issue
59	getRoleAdmin	read	Passed	No Issue
60	grantRole	write	Role Admin	No Issue
61	revokeRole	write	role admin	No Issue
62	renounceRole	write	Passed	No Issue
63	_setupRole	internal	Passed	No Issue
64	_setRoleAdmin	internal	Passed	No Issue

65	_grantRole	internal	Passed	No Issue
66	_revokeRole	internal	Passed	No Issue
67	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
68	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
69	onlyProxy	modifier	Passed	No Issue
70	notDelegated	modifier	Passed	No Issue
71	proxiableUUID	external	Passed	No Issue
72	upgradeTo	external	access only Proxy	No Issue
73	upgradeToAndCall	external	access only Proxy	No Issue
74	_authorizeUpgrade	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] Visibility can be external over the public:

Description:

Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

[I-02] Unused state variable:

```
uint256 private DELETED_1; // not used (liquidityIndex)
uint256 private DELETED_2; // not used (liquidityIndexDenominator)
```

Description:

DELETED_1 & DELETED_2 are defined but not used in the code.

Recommendation: We suggest removing unused variables.

[I-03] Missing-zero-address-validation:

```
function assetToCredit(address owner, uint256 amount) public view returns(uint256 credit) {
    if (amount > MAX_SUPPLY / 10 ** 45) {
        return MAX_SUPPLY;
    }
    uint256 amountRay = WadRayMath.wadToRay(amount);
    uint256 creditsPerTokenRay = WadRayMath.wadToRay(_creditsPerToken(owner));
    uint256 creditRay = WadRayMath.rayMul(amountRay, creditsPerTokenRay);
    return WadRayMath.rayToWad(creditRay);
}
```

Description:

Detect missing zero address validation.

- assetToCredit()
- creditToAsset()
- subCredits()

without specifying the owner loses ownership of the contract.

Recommendation: We suggest first checking that the address is not zero.

[I-04] Language Specific:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;
```

Description:

The contract has an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation: We suggest that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

[I-05] Compile time warnings:

Description:

Warning: This declaration shadows an existing declaration.

100 | function initialize(string calldata name, string calldata symbol, uint8 decimals) initializer public {

178 | function name() public view returns (string memory) {

186 | function symbol() public view returns (string memory)

Recommendation: We suggest if a variable is declared in the function then no need to declare it again in the same function. so we need to remove unwanted declarations.

Warning: Function state mutability can be restricted to view.

660 | function _isNonRebasingAccount(address _account) internal returns (bool)

Recommendation: We suggest state mutability.

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 Owner functions:

UsdPlusToken.sol

- _authorizeUpgrade: Admin roles upgrade by default admin.
- setExchanger: The exchanger address can be set by the admin.
- setPayoutManager: The Payout Manager address can be set by the admin.
- setRoleManager: The Manager address can be set by the admin.
- pause: Pause status set true by the Portfolio Agent.
- unpause: Unpause status set true by the Portfolio Agent.
- mint: Mints new tokens, increasing total supply only by the Exchanger.
- burn: Burns tokens, decreasing total Supply only by the only Exchanger.
- changeSupply: Modify the supply without minting new tokens only by the Exchanger.

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 5 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".**

Our Methodology

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

Manual Code Review:

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

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and 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).

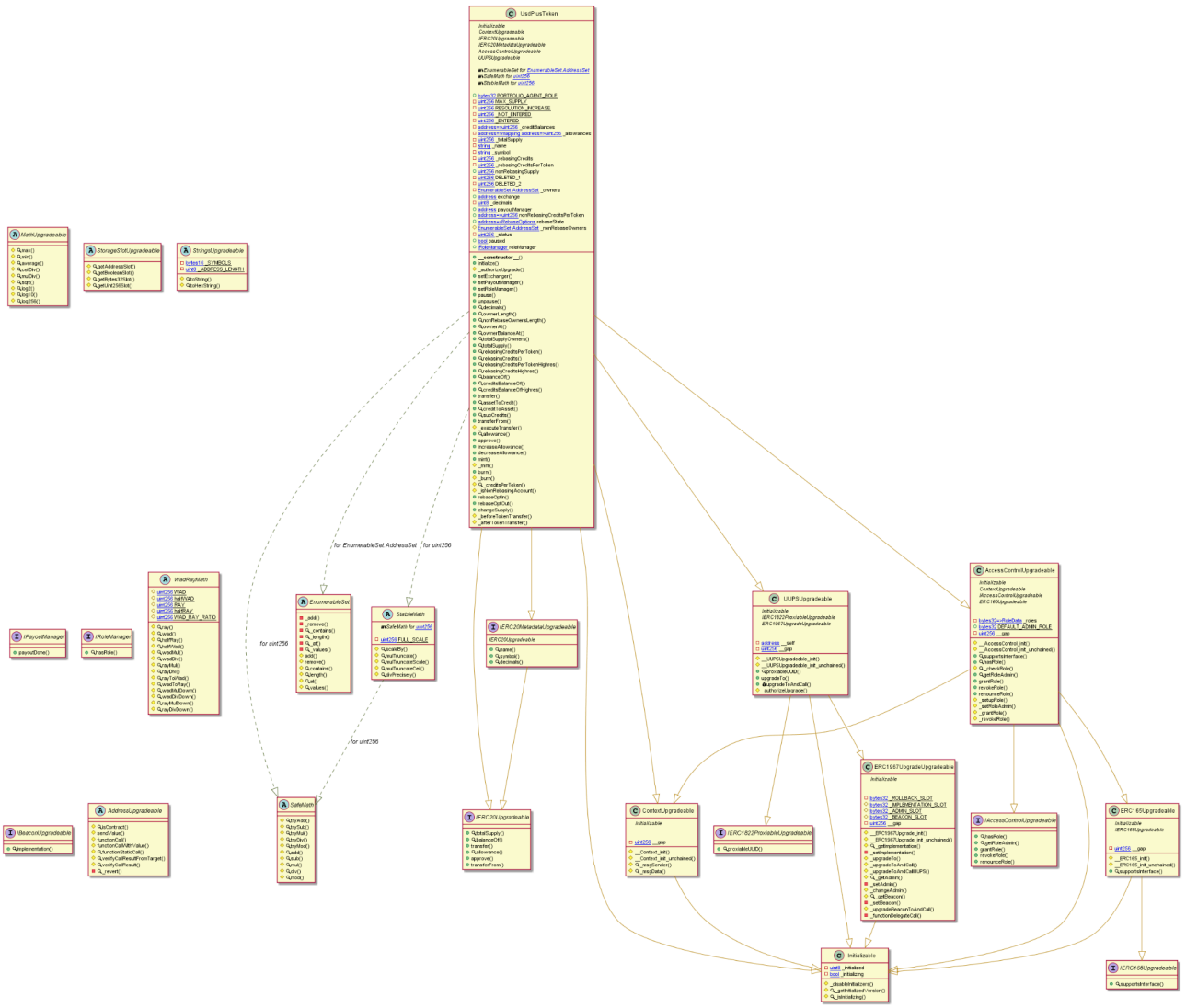
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



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.

UsdPlusToken.sol

```
INFO:Detectors:
UsdPlusToken.initialize(string,string,uint8).name (UsdPlusToken.sol#1334) shadows:
	- IERC20MetadataUpgradeable.name() (UsdPlusToken.sol#929) (function)
UsdPlusToken.initialize(string,string,uint8).symbol (UsdPlusToken.sol#1334) shadows:
	- IERC20MetadataUpgradeable.symbol() (UsdPlusToken.sol#931) (function)
UsdPlusToken.initialize(string,string,uint8).decimals (UsdPlusToken.sol#1334) shadows:
	- UsdPlusToken.decimals() (UsdPlusToken.sol#1406-1408) (function)
	- IERC20MetadataUpgradeable.decimals() (UsdPlusToken.sol#933) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
UsdPlusToken.totalSupplyOwners() (UsdPlusToken.sol#1426-1436) has external calls inside a
loop: total += this.balanceOf(_owners.at(index)) (UsdPlusToken.sol#1432)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Pragma version^0.8.0 (UsdPlusToken.sol#2) allows old versions
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Constant WadRayMath.halfWAD (UsdPlusToken.sol#695) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant WadRayMath.halfRAY (UsdPlusToken.sol#698) is not in
UPPER_CASE_WITH_UNDERSCORES
Parameter UsdPlusToken.setExchanger(address)._exchanger (UsdPlusToken.sol#1379) is not in
mixedCase
Parameter UsdPlusToken.setPayoutManager(address)._payoutManager
(UsdPlusToken.sol#1385) is not in mixedCase
Parameter UsdPlusToken.setRoleManager(address)._roleManager (UsdPlusToken.sol#1391) is
not in mixedCase
Parameter UsdPlusToken.balanceOf(address)._account (UsdPlusToken.sol#1458) is not in
```

```
mixedCase
Parameter UsdPlusToken.creditsBalanceOf(address)._account (UsdPlusToken.sol#1467) is not in
mixedCase
Parameter UsdPlusToken.creditsBalanceOfHighres(address)._account (UsdPlusToken.sol#1483)
is not in mixedCase
Parameter UsdPlusToken.transfer(address,uint256)._to (UsdPlusToken.sol#1497) is not in
mixedCase
Parameter UsdPlusToken.transfer(address,uint256)._value (UsdPlusToken.sol#1497) is not in
mixedCase
Parameter UsdPlusToken.transferFrom(address,address,uint256)._from
(UsdPlusToken.sol#1547) is not in mixedCase
Parameter UsdPlusToken.transferFrom(address,address,uint256)._to (UsdPlusToken.sol#1548) is
not in mixedCase
Parameter UsdPlusToken.transferFrom(address,address,uint256)._value
(UsdPlusToken.sol#1549) is not in mixedCase
Parameter UsdPlusToken.allowance(address,address)._owner (UsdPlusToken.sol#1593) is not in
mixedCase
Parameter UsdPlusToken.allowance(address,address)._spender (UsdPlusToken.sol#1593) is not
in mixedCase
Parameter UsdPlusToken.approve(address,uint256)._spender (UsdPlusToken.sol#1604) is not in
mixedCase
Parameter UsdPlusToken.approve(address,uint256)._value (UsdPlusToken.sol#1604) is not in
mixedCase
Parameter UsdPlusToken.increaseAllowance(address,uint256)._spender
(UsdPlusToken.sol#1616) is not in mixedCase
Parameter UsdPlusToken.increaseAllowance(address,uint256)._addedValue
(UsdPlusToken.sol#1616) is not in mixedCase
Parameter UsdPlusToken.decreaseAllowance(address,uint256)._spender
(UsdPlusToken.sol#1628) is not in mixedCase
Parameter UsdPlusToken.decreaseAllowance(address,uint256)._subtractedValue
(UsdPlusToken.sol#1628) is not in mixedCase
Parameter UsdPlusToken.mint(address,uint256)._account (UsdPlusToken.sol#1639) is not in
mixedCase
Parameter UsdPlusToken.mint(address,uint256)._amount (UsdPlusToken.sol#1639) is not in
mixedCase
Parameter UsdPlusToken.rebaseOptIn(address)._address (UsdPlusToken.sol#1714) is not in
mixedCase
Parameter UsdPlusToken.rebaseOptOut(address)._address (UsdPlusToken.sol#1734) is not in
mixedCase
Parameter UsdPlusToken.changeSupply(uint256)._newTotalSupply (UsdPlusToken.sol#1747) is
not in mixedCase
Variable UsdPlusToken.DELETED_1 (UsdPlusToken.sol#1289) is not in mixedCase
Variable UsdPlusToken.DELETED_2 (UsdPlusToken.sol#1290) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
UsdPlusToken (UsdPlusToken.sol#1263-1842) does not implement functions:
        - IERC20MetadataUpgradeable.name() (UsdPlusToken.sol#929)
```

```
        - IERC20MetadataUpgradeable.symbol() (UsdPlusToken.sol#931)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Detectors:
UsdPlusToken.DELETED_1 (UsdPlusToken.sol#1289) is never used in UsdPlusToken
(UsdPlusToken.sol#1263-1842)
UsdPlusToken.DELETED_2 (UsdPlusToken.sol#1290) is never used in UsdPlusToken
(UsdPlusToken.sol#1263-1842)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
UsdPlusToken.DELETED_1 (UsdPlusToken.sol#1289) should be constant
UsdPlusToken.DELETED_2 (UsdPlusToken.sol#1290) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
The function UsdPlusToken.totalSupplyOwners() (UsdPlusToken.sol#1426-1436) reads total +=
this.balanceOf(_owners.at(index)) (UsdPlusToken.sol#1432) with `this` which adds an extra
STATICCALL.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-variable-read-in-external-context
INFO:Slither:UsdPlusToken.sol analyzed (23 contracts with 93 detectors), 172 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.

UsdPlusToken.sol

```
Gas costs:
Gas requirement of function UsdPlusToken.pause 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: 473:4:

Gas costs:
Gas requirement of function UsdPlusToken.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: 477:4:

This on local calls:
Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.
Pos: 534:21:

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 in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 1066:8:

Similar variable names:
UsdPlusToken.creditToAsset(address,uint256) : Variables have very similar names "_owners" and "owner". Note: Modifiers are currently not considered by this static analysis.
Pos: 687:74:

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: 1059:8:

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: 995:8:

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.

UsdPlusToken.sol

```
Compiler version >=0.8.0 <0.9.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:1
global import of path @openzeppelin/contracts/utils/structs/EnumerableSet.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:3
global import of path @openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:4
global import of path @openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:5
global import of path @openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:6
global import of path @openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:7
global import of path @openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:8
Constant name must be in capitalized SNAKE_CASE
Pos: 5:26
Constant name must be in capitalized SNAKE_CASE
Pos: 5:29
Error message for require is too long
Pos: 9:72
Error message for require is too long
```

```
Pos: 9:88
Error message for require is too long
Pos: 9:102
Error message for require is too long
Pos: 9:118
Error message for require is too long
Pos: 9:141
Error message for require is too long
Pos: 9:155
Error message for require is too long
Pos: 9:170
Error message for require is too long
Pos: 9:184
Error message for require is too long
Pos: 9:199
Contract has 20 states declarations but allowed no more than 15
Pos: 1:333
Variable name must be in mixedCase
Pos: 5:359
Variable name must be in mixedCase
Pos: 5:360
Explicitly mark visibility of state
Pos: 5:370
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:400
Visibility modifier must be first in list of modifiers
Pos: 99:405
Code contains empty blocks
Pos: 5:423
Use double quotes for string literals
Pos: 43:454
Use double quotes for string literals
Pos: 47:460
Use double quotes for string literals
Pos: 45:466
Use double quotes for string literals
Pos: 50:1033
Code contains empty blocks
Pos: 16:1086
```

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