Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: USDC (anyUSDC)
Website: multichain.org
Platform: Binance Network
Language: Solidity
Date: April 14th, 2025

Table of Contents

Email: audit@EtherAuthority.io

THIS IS A SECURITY AUDIT REPORT DOCUMENT THAT MAY CONTAIN INFORMATION THAT 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 contract audit initiatives, the smart contract of the USDC (anyUSDC) Token from multichain.org was audited. The audit was performed using manual analysis and automated software tools. This report presents all the findings regarding the audit performed on April 14th, 2025.

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

The AnyswapV5ERC20 contract is a specialized ERC20 token implementation used in cross-chain token bridging scenarios. It extends from IAnyswapV3ERC20 and introduces enhanced control, security, and functionality for managing token minting, burning, and transferring across different chains. Here's a high-level description of its features and design:

Purpose:

A cross-chain compatible ERC20 token contract designed for the Anyswap/Multichain ecosystem. It provides vault-based minting/burning logic, off-chain signature approvals (EIP-2612), and a customizable time-lock mechanism for managing key roles.

Audit scope

Name	**Code Review and Security Analysis Report for USDC (anyUSDC) Token Smart Contract**
Platform	**Binance Network**
File	AnyswapV5ERC20.sol
File Smart Contract Code	0x8965349fb649a33a30cbfda057d8ec2c48abe2a2
Audit Date	April 14th, 2025

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: USDC • Symbol: anyUSDC	**YES, This is valid.**
Key Features: • The AnyswapV5ERC20 contract is a cross-chain ERC20 wrapper token implementation used in the Multichain (Anyswap) protocol. It combines standard ERC20 functionality with advanced features for bridging, minting, burning, and cross-chain interoperability. **Access Control and Authorization:** • **onlyAuth**: Restricts functions to authorized minters. • **onlyVault**: Restricts functions to the current MPC owner (mpc()), which may include a timelocked vault address.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contract is **"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, 1 low, and 3 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	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	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**

Code Quality

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

The libraries in USDC (anyUSDC) 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 anyUSDC 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 contract. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an anyUSDC Token smart contract code in the form of a BSCscan 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 that is 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

AnyswapV5ERC20.sol : Functions

Sl.	Functions	Type	Observation	Conclusion
1	onlyAuth	modifier	Passed	No Issue
2	onlyVault	modifier	Passed	No Issue
3	owner	read	Passed	No Issue
4	mpc	read	Passed	No Issue
5	setVaultOnly	external	access only Vault	No Issue
6	initVault	external	access only Vault	No Issue
7	setMinter	external	access only Vault	No Issue
8	setVault	external	Missing Event for Critical Functions	Refer Audit Findings
9	applyVault	external	access only Vault	No Issue
10	applyMinter	external	access only Vault	No Issue
11	revokeMinter	external	access only Vault	No Issue
12	getAllMinters	external	Passed	No Issue
13	changeVault	external	access only Vault	No Issue
14	changeMPCOwner	write	access only Vault	No Issue
15	mint	external	Unrestricted `mint` Access	Refer Audit Findings
16	burn	external	access only Auth	No Issue
17	Swapin	write	access only Auth	No Issue
18	Swapout	write	Passed	No Issue
19	constructor	write	Passed	No Issue
20	totalSupply	external	Passed	No Issue
21	depositWithPermit	external	Passed	No Issue
22	depositWithTransferPermit	external	Passed	No Issue
23	deposit	external	Passed	No Issue
24	deposit	external	Passed	No Issue
25	deposit	external	Passed	No Issue
26	depositVault	external	access only Vault	No Issue
27	_deposit	internal	Passed	No Issue
28	withdraw	external	Passed	No Issue
29	withdraw	external	Passed	No Issue
30	withdraw	external	Passed	No Issue
31	_withdraw	internal	Passed	No Issue
32	_mint	internal	Passed	No Issue
33	_burn	internal	Passed	No Issue
34	approve	external	Passed	No Issue
35	approveAndCall	external	Passed	No Issue
36	permit	external	Redundant/Unused Functions	Refer Audit Findings
37	transferWithPermit	external	Passed	No Issue
38	verifyEIP712	internal	Passed	No Issue

39	verifyPersonalSign	internal	Passed	No Issue
40	prefixed	internal	Passed	No Issue
41	transfer	external	Passed	No Issue
42	transferFrom	external	Passed	No Issue
43	transferAndCall	external	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 being lost
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, which 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.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Missing Event for Critical Functions:

Functions like `setVault` lack event emissions, making on-chain activity harder to monitor.

Resolution: Emit events for all state-changing functions to improve transparency and auditing.

Very Low / Informational / Best practices:

(1) Unrestricted `mint` Access:

The `mint` function can be called by any address if `vault` is not set (`address(0)`), potentially allowing unlimited token minting.

Resolution: Ensure `vault` is initialized during deployment and make `mint` callable only by the vault. Add a check that reverts if `vault == address(0)`.

(2) Redundant/Unused Functions:

The `permit` function implementation may conflict with ERC20 extensions if not implemented properly across all frontends or wallets.

Resolution: Ensure frontend support is available or provide documentation for using permit-based approvals.

(3) Old Solidity Version:

Contract uses `pragma solidity ^0.8.2;`. While secure, newer versions may provide security improvements.

Resolution: Consider upgrading to the latest stable version (e.g., `^0.8.30`) for better security and features.

Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key is compromised, then it creates trouble. The following are Admin functions:

AnyswapV5ERC20.sol

- initVault(address _vault): Initializes the vault (used for CREATE2 compatibility) by the vault.
- setVaultOnly(bool enabled): Enables/disables swapout to force usage of the vault. burn by the vault.
- setVault: Sets a new vault address with a delay by the vault.
- setMinter(address): Proposes a new minter by the vault.
- applyMinter(): Finalizes the minter addition after delay by the vault.
- applyVault: Finalizes the vault addition after delay by the vault.
- revokeMinter(address): Instantly revokes a minter by the vault.
- changeVault() / changeMPCOwner(): Initiates ownership transfer to new MPC or vault.
- mint(address, uint256): Mints tokens to a user (by an authorized minter).
- burn(address, uint256): Burns tokens from a user (by an authorized minter).
- Swapin(bytes32, address, uint256): Mint tokens on swap-in(by an authorized minter).

Conclusion

We were given a contract code in the form of bscscan web links. We have used all possible tests based on the given objects as files. We observed 1 low and 3 informational issues in the smart contract, and those issues are not critical. So, **it's good to go for production**.

Since possible test cases can be unlimited for such smart contract protocols, 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 whitebox 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 not to 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 under the best industry practices at the date of this report, concerning: 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).

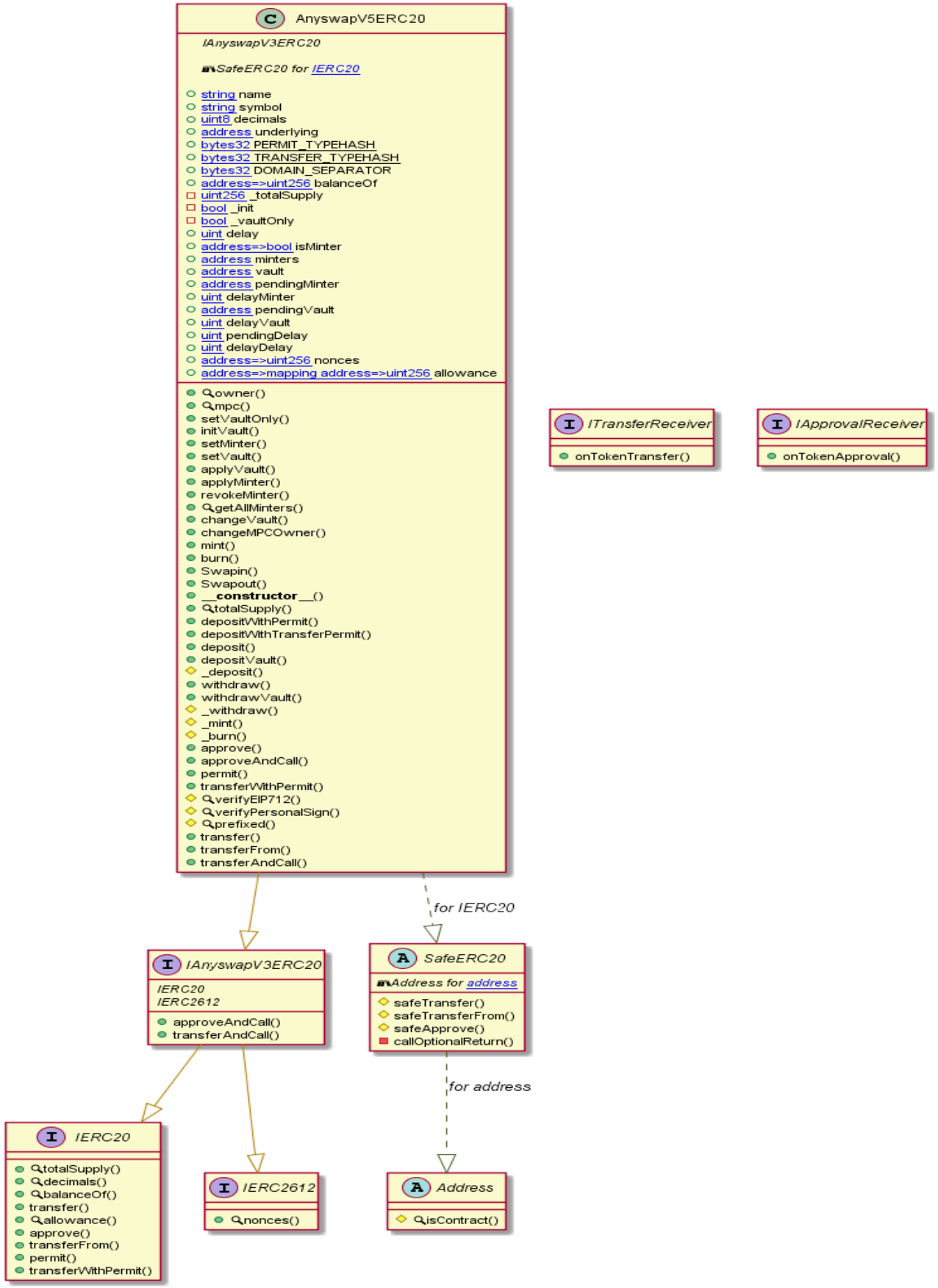
Since the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done 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 - anyUSDC Token



Email: audit@EtherAuthority.io

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 >> AnyswapV5ERC20.sol

```
INFO:Detectors:
AnyswapV5ERC20.depositWithPermit(address,uint256,uint256,uint8,bytes32,bytes32,address)
(AnyswapV5ERC20.sol#337-341) uses arbitrary from in transferFrom in combination with
permit: IERC20(underlying).safeTransferFrom(target,address(this),value)
(AnyswapV5ERC20.sol#339)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom-used-with-permit
INFO:Detectors:
AnyswapV5ERC20.depositWithTransferPermit(address,uint256,uint256,uint8,bytes32,bytes32,address) (AnyswapV5ERC20.sol#343-346) ignores return value by
IERC20(underlying).transferWithPermit(target,address(this),value,deadline,v,r,s)
(AnyswapV5ERC20.sol#344)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
AnyswapV5ERC20.initVault(address)._vault (AnyswapV5ERC20.sol#207) lacks a zero-check on
:
        - vault = _vault (AnyswapV5ERC20.sol#209)
        - pendingVault = _vault (AnyswapV5ERC20.sol#210)
AnyswapV5ERC20.setMinter(address)._auth (AnyswapV5ERC20.sol#217) lacks a zero-check on
:
        - pendingMinter = _auth (AnyswapV5ERC20.sol#218)
        - pendingVault = _vault (AnyswapV5ERC20.sol#318)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
AnyswapV5ERC20.constructor(string,string,uint8,address,address)
(AnyswapV5ERC20.sol#302-330) uses assembly
    - INLINE ASM (AnyswapV5ERC20.sol#322)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
```

```
Function AnyswapV5ERC20.Swapin(bytes32,address,uint256)
(AnyswapV5ERC20.sol#275-279) is not in mixedCase
Function AnyswapV5ERC20.Swapout(uint256,address) (AnyswapV5ERC20.sol#281-287) is not
in mixedCase
Variable AnyswapV5ERC20.DOMAIN_SEPARATOR (AnyswapV5ERC20.sol#149) is not in
mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
AnyswapV5ERC20.delay (AnyswapV5ERC20.sol#162) should be constant
AnyswapV5ERC20.delayDelay (AnyswapV5ERC20.sol#179) should be constant
AnyswapV5ERC20.pendingDelay (AnyswapV5ERC20.sol#178) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decl
ared-constant
INFO:Slither:AnyswapV5ERC20.sol analyzed (8 contracts with 93 detectors), 36 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.

AnyswapV5ERC20.sol

```
Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in AnyswapV5ERC20.deposit(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
Pos: 348:4:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 485:16:

Gas costs:
Gas requirement of function AnyswapV5ERC20.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 264:4:

Gas costs:
Gas requirement of function AnyswapV5ERC20.depositWithPermit 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: 337:4:

Similar variable names:
AnyswapV5ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 409:30:

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: 591:8:
```

Solhint Linter

Solhint 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.

AnyswapV5ERC20.sol

```
Compiler version 0.8.2 does not satisfy the ^0.5.8 semver requirement
Pos: 1:26
Error message for require is too long
Pos: 9:119
Avoid making time-based decisions in your business logic
Pos: 22:258
Function name must be in mixedCase
Pos: 5:274
Function name must be in mixedCase
Pos: 5:280
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:301
Provide an error message for require
Pos: 13:307
Avoid making time-based decisions in your business logic
Pos: 22:318
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:321
Provide an error message for require
Pos: 9:368
Error message for require is too long
Pos: 9:593
```

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