Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Multichain BTC
Website: app.multichain.org
Platform: Ethereum
Language: Solidity
Date: March 28th, 2026

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 smart contracts of Multichain BTC from app.multichain.org were audited. 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 March 28th, 2026.

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

AnyswapV6ERC20 is a cross-chain compatible ERC20 token contract designed for bridging assets between multiple blockchains. It enables seamless **minting, burning, deposit, withdrawal, and swap operations** through a controlled **vault-based architecture**.

The contract supports both:

- **Wrapped tokens** (backed by an underlying asset), and
- **Mintable tokens** (for cross-chain liquidity)

It is commonly used in cross-chain protocols (like Multichain/Anyswap) where tokens are locked on one chain and minted on another.

The design includes:

- Role-based minting (minters)
- Vault-controlled governance
- Timelock-protected critical operations
- Safe ERC20 handling using low-level call protection

Audit scope

Name	Code Review and Security Analysis Report for Multichain BTC Smart Contract
Platform	**Ethereum**
Language	**Solidity**
File	AnyswapV6ERC20.sol
Smart Contract Code	0x66eFF5221ca926636224650Fd3B9c497FF828F7D
Audit Date	March 28th, 2026

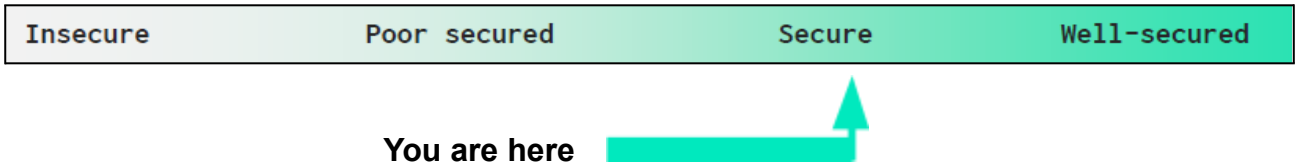
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Key Features of the AnyswapV6ERC20 contract: • Vault-controlled ownership and admin functions • Multi-minter role for minting and burning • Cross-chain swap support (Swapin / Swapout) • Supports underlying token (lock/unlock or mint/burn model) • Deposit & withdraw functionality • 2-day timelock for critical changes • Emergency minter revoke • Safe ERC20 operations (Secure transfers) • Vault-only mode restriction option • Standard ERC20 functionality (transfer, approve, etc.)	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity 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 critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 3 very low-level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	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	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

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

The libraries in Multichain BTC 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 Multichain BTC.

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 not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Multichain BTC smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but 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	onlyAuth	modifier	Passed	No Issue
3	onlyVault	modifier	Passed	No Issue
4	owner	external	Passed	No Issue
5	mpc	external	Passed	No Issue
6	setVaultOnly	external	access only Vault	No Issue
7	initVault	external	access only Vault	No Issue
8	setVault	external	access only Vault	No Issue
9	applyVault	external	access only Vault	No Issue
10	setMinter	external	Lack of Event Emission for Critical State Changes	Refer Audit Findings
11	applyMinter	external	Lack of Event Emission for Critical State Changes	Refer Audit Findings
12	revokeMinter	external	Lack of Event Emission for Critical State Changes	Refer Audit Findings
13	getAllMinters	external	Passed	No Issue
14	changeVault	external	access only Vault	No Issue
15	mint	external	access only Auth	No Issue
16	burn	external	access only Auth	No Issue
17	Swapin	external	access only Auth	No Issue
18	Swapout	external	Passed	No Issue
19	verifyBindAddr	internal	Passed	No Issue
20	totalSupply	external	Passed	No Issue
21	deposit	external	Deposit() Without Explicit Amount	Refer Audit Findings
22	deposit	external	Passed	No Issue
23	deposit	external	Passed	No Issue
24	depositVault	external	access only Vault	No Issue
25	_deposit	internal	Passed	No Issue
26	withdraw	external	Passed	No Issue
27	withdraw	external	Passed	No Issue
28	withdraw	external	Passed	No Issue
29	withdrawVault	external	access only Vault	No Issue
30	_withdraw	internal	Passed	No Issue
31	_mint	internal	Passed	No Issue
32	_burn	internal	Passed	No Issue
33	approve	external	Passed	No Issue
34	transfer	external	Passed	No Issue
35	transferFrom	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 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) Use the latest solidity version:

```
pragma solidity ^0.8.2;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use the latest solidity versions.

(2) Deposit() Without Explicit Amount:
function deposit()
Deposits the entire balance of the caller:
uint _amount = IERC20(underlying).balanceOf(msg.sender);
This can lead to unexpected token transfers.

Resolution: Remove this function OR
Require explicit user confirmation with amount parameter

(3) Lack of Event Emission for Critical State Changes:

Some important operations lack sufficient event logging:

- setMinter
- applyMinter
- revokeMinter

Resolution: Emit events for all governance changes.

Centralization Risk

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:

AnyswapV6ERC20.sol

- setVaultOnly: The Vault can enable/disable swapout restriction mode.
- initVault: The Vault can initialize a vault once.
- setVault: Proposes a new vault with a timelock delay by vault.
- applyVault: Applies the pending vault change after delay by vault.
- setMinter: Proposes a new minter with delay by vault.
- applyMinter: Approves and adds the pending minter after delay by vault.
- revokeMinter: Immediately removes a minter (emergency) by vault.
- changeVault: Instantly updates vault without delay (emergency/admin) by vault.
- mint: Mints tokens to a user (only authorized minters).
- burn: Burns tokens from a user (only authorized minters).
- Swapin: Mints or transfers tokens during cross-chain inbound swap(only authorized minters).
- depositVault: Vault deposits tokens without transfer (admin use).
- withdrawVault: Vault withdraws tokens from users.

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 3 informational issues in the smart contracts. And those issues 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 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).

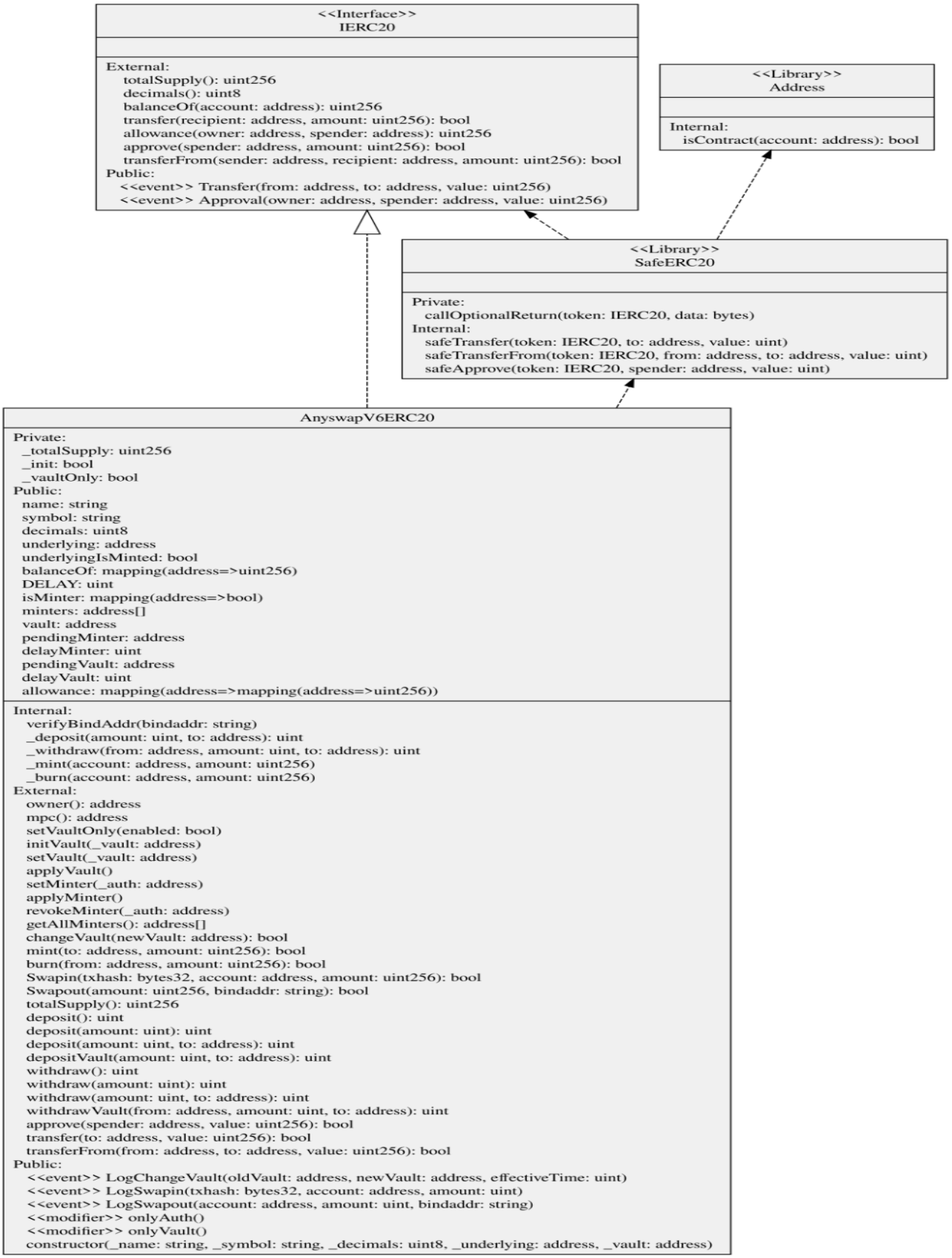
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



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 >> AnyswapV6ERC20.sol

```
INFO:Detectors:
AnyswapV6ERC20 uses Solidity version ^0.8.2 (AnyswapV6ERC20.sol#3)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect
-solc-version
AnyswapV6ERC20.Swapin(bytes32,address,uint256)
(AnyswapV6ERC20.sol#~120-132) allows arbitrary token minting via
authorized minters
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary
-mint
AnyswapV6ERC20.mint(address,uint256) (AnyswapV6ERC20.sol#~102-106)
allows minting without supply cap
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unbounded
-minting
AnyswapV6ERC20.burn(address,uint256) (AnyswapV6ERC20.sol#~108-112)
allows arbitrary burning by authorized minters
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary
-burn
AnyswapV6ERC20.Swapout(uint256,string) (AnyswapV6ERC20.sol#~134-146)
burns tokens based on user balance without additional validation
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked
-burn
AnyswapV6ERC20.verifyBindAddr(string) (AnyswapV6ERC20.sol#~148-151)
performs weak validation (only length check)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#weak-prng
AnyswapV6ERC20.deposit() (AnyswapV6ERC20.sol#~173-177) transfers full
user balance instead of specified amount
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous
-usage-of-balance
AnyswapV6ERC20.changeVault(address) (AnyswapV6ERC20.sol#~88-96) allows
immediate privilege change without timelock
```

```
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-t
imelock
AnyswapV6ERC20.setMinter(address) / applyMinter()
(AnyswapV6ERC20.sol#~66-80) uses delayed authorization but lacks event
emission for transparency
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-e
vents-access-control
AnyswapV6ERC20.revokeMinter(address) (AnyswapV6ERC20.sol#~82-85) allows
immediate privilege revocation without delay
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#centraliz
ation-risk
AnyswapV6ERC20.transfer(address,uint256) (AnyswapV6ERC20.sol#~230-242)
missing SafeMath but relies on Solidity ^0.8 overflow checks
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arithmeti
c
AnyswapV6ERC20.transferFrom(address,address,uint256)
(AnyswapV6ERC20.sol#~244-268) uses max allowance pattern which may
introduce approval race conditions
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#erc20-int
erface
AnyswapV6ERC20 uses multiple privileged roles:
vault
isMinter mapping
(AnyswapV6ERC20.sol#~40-50)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#centraliz
ation-risk
AnyswapV6ERC20 naming convention deviation detected:
Swapin
Swapout
(AnyswapV6ERC20.sol#~120,134)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#naming-co
nvention
INFO:Slither:AnyswapV6ERC20.sol analyzed (3 contracts), 14 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.

AnyswapV6ERC20.sol

```
Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in
AnyswapV6ERC20.deposit(): Could potentially lead to re-entrancy
vulnerability. Note: Modifiers are currently not considered by this
static analysis.
Pos: 239:4:

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: 25:8:

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: 166:45:

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to
unexpected behavior if the return value is not handled properly.
Please use Direct Calls via specifying the called contract's
interface.
Pos: 51:50:

Gas costs:
Gas requirement of function AnyswapV6ERC20.deposit 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: 239:4:

ERC20:
ERC20 contract's "decimals" function should have "uint8" as return
type
Pos: 10:4:

Similar variable names:
AnyswapV6ERC20._mint(address,uint256) : Variables have very similar
names "account" and "amount". Note: Modifiers are currently not
```

```
considered by this static analysis.
Pos: 304:43:

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: 300:8:
```

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.

AnyswapV6ERC20.sol

```
Found more than One contract per file. 4 contracts found!
Pos: 1:2
Rule is set with explicit type [var/s: uint]
Pos: 53:32
Error message for require is too long: 54 counted / 32 allowed
Pos: 9:41
Use Custom Errors instead of require statements
Pos: 9:41
Use Custom Errors instead of require statements
Pos: 9:47
Use Custom Errors instead of require statements
Pos: 9:51
Error message for require is too long: 42 counted / 32 allowed
Pos: 13:55
Use Custom Errors instead of require statements
Pos: 13:55
Immutable variables name are set to be in capitalized SNAKE_CASE
Pos: 5:64
```

Email: audit@EtherAuthority.io