www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: ANY Ethereum
Platform: Binance Smart Chain
Language: Solidity
Date: January 5th, 2026

Table of contents

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.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of ANY Ethereum (anyETH) from anyswap.exchange 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 January 5th, 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

1. **Token Type**
   This project is an **ERC20-compatible token** built on the BNB Smart Chain, enabling standard token transfers and balances.
2. **Core Standard**
   It follows the OpenZeppelin ERC20 framework, ensuring widely accepted token behavior and security practices.
3. **Swap Mechanism**
   The contract introduces Swapin and Swapout functions for **cross-chain asset transfers**.
4. **Bridge Functionality**
   It acts as a **bridge token**, allowing assets to move between different blockchains via a centralized authority.
5. **Minting Capability**
   Tokens are minted through the Swapin function, which is controlled by the contract owner.
6. **Burn Mechanism**
   Tokens are burned using the Swapout function when users transfer assets out of the chain.
7. **Ownership Control**
   The contract includes an onlyOwner modifier, giving administrative control to a single owner address.

8. **Delayed Ownership Transfer**

   Ownership changes are not immediate—they activate after a specific number of blocks, adding a delay mechanism.

9. **Security Measures**

   Uses SafeMath to prevent overflow/underflow issues in arithmetic operations.

10. **Centralized Trust Model**

    The system relies on a trusted owner for minting and bridging, meaning it is **not fully decentralized**.

Audit scope

Name	**Code Review and Security Analysis Report for ANY Ethereum (anyETH) Smart Contract**
Platform	**Binance Smart Chain**
Language	**Solidity**
File	Erc20SwapAsset.sol
Smart Contract Code	0x6f817a0ce8f7640add3bc0c1c2298635043c2423
Audit Date	January 5th, 2026

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Key features: • **Wrapped BTC token (anyBTC)** with 8 decimals • **Swapin (mint)** by owner when BTC is deposited • **Swapout (burn)** by users to withdraw BTC • **Owner-only minting** (centralized control) • **Delayed ownership transfer** (~2 days) • **Basic BTC address validation** • **Event logging** for swaps and ownership changes	**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 3 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	Moderated
	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

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 ANY Ethereum 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 ANY Ethereum.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given ANY Ethereum 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 are used in this smart contract infrastructure that 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	onlyOwner	modifier	Passed	No Issue
3	constructor	write	Passed	No Issue
4	owner	read	Passed	No Issue
5	changeDCRMOwner	write	access only Owner	No Issue
6	Swapin	write	Centralized Minting via Swapin()	Refer Audit Findings
7	Swapout	write	Passed	No Issue
8	name	read	Passed	No Issue
9	symbol	read	Passed	No Issue
10	decimals	read	Passed	No Issue
11	totalSupply	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	transfer	write	Passed	No Issue
14	allowance	read	Passed	No Issue
15	approve	write	Passed	No Issue
16	transferFrom	write	Passed	No Issue
17	increaseAllowance	write	Passed	No Issue
18	decreaseAllowance	write	Passed	No Issue
19	_transfer	internal	Passed	No Issue
20	_mint	internal	Passed	No Issue
21	_burn	internal	Passed	No Issue
22	_approve	internal	Passed	No Issue
23	_burnFrom	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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 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

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Centralized Minting via Swapin()

Description:

The contract allows minting of tokens through the Swapin() function, which is typically controlled by the owner (DCRM operator). There is no decentralized validation mechanism enforced on-chain.

Recommendation:

- Introduce multi-signature control for minting
- Add rate limits or mint caps
- Emit stronger validation checks for cross-chain proofs

(2) Lack of Maximum Supply Cap

Description:

There is no maxSupply limit enforced in the contract.

Impact:

- Unlimited token minting possible via Swapin()
- Token economics can be manipulated

Recommendation:

- Introduce immutable maxSupply
- Enforce check inside mint logic

(3) Outdated Solidity Version

Description:
Contract uses Solidity v0.5.4

Impact:

- Missing built-in overflow checks (pre-0.8.x)
- Higher risk of legacy vulnerabilities

Recommendation:

- Upgrade to Solidity ≥0.8.x
- Remove SafeMath (built-in checks available)

Centralization

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

Erc20SwapAsset.sol

- changeDCRMOwner: Allows the current owner to initiate ownership transfer with a delayed activation mechanism.
- Swapin: Allows the owner to mint tokens to a user based on cross-chain deposit verification.

Conclusion

We were given a contract code in the form of bscscan web links. And we have used all possible tests based on given objects as files. We observed 3 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.

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 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, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

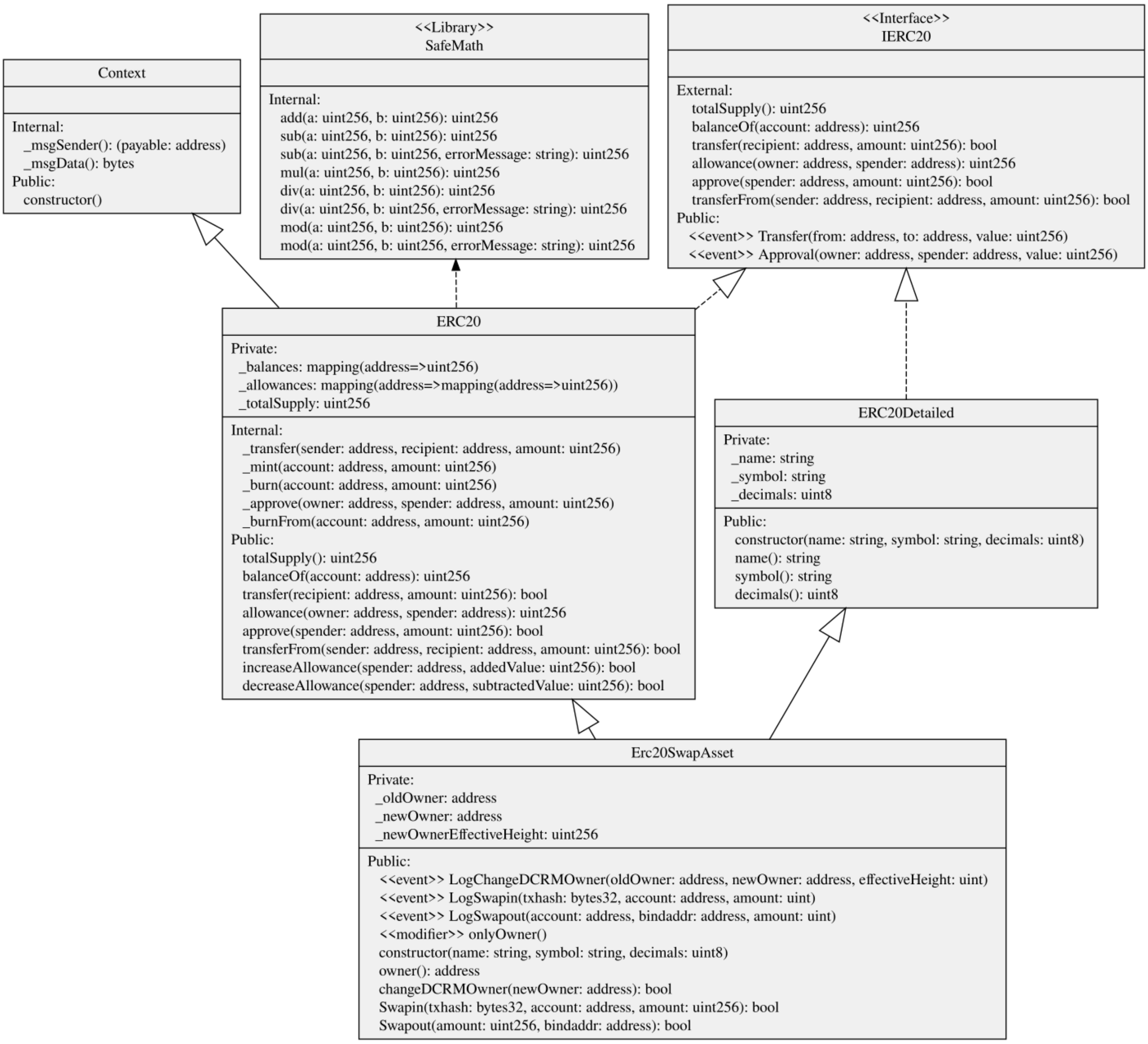
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - ANY Ethereum (anyETH)



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 >> Erc20SwapAsset.sol

```
INFO:Detectors:
Erc20SwapAsset.Swapin(bytes32,address,uint256)
(Erc20SwapAsset.sol#43-48) allows arbitrary token minting
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arbitra
ry-mint
Erc20SwapAssetuses Solidity version ^0.5.0 (Erc20SwapAsset.sol#1)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorre
ct-solc-version
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#depreca
ted-standards
Erc20SwapAsset.Swapin(bytes32,address,uint256)
(Erc20SwapAsset.sol#43-48) missing zero-address validation
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing
-zero-address-validation
Erc20SwapAsset.changeDCRMOwner(address) (Erc20SwapAsset.sol#34-41)
uses block.number for control logic
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#block-n
umber

Erc20SwapAssetowner logic uses multiple state variables:
- _oldOwner
- _newOwner
- _newOwnerEffectiveHeight
(Erc20SwapAsset.sol#24-27)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#shadowi
ng-state

Erc20SwapAsset.Swapout(uint256,string) (Erc20SwapAsset.sol#50-55)
burns tokens without additional validation
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uncheck
ed-burn
```


Email: audit@EtherAuthority.io

```
Erc20SwapAssetnaming convention deviation detected:
- Swapin
- Swapout
(Erc20SwapAsset.sol#43,50)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#naming-
convention

INFO:Slither:Erc20SwapAsset.sol analyzed (6 contracts), 9 result(s)
found
```


Email: audit@EtherAuthority.io

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.

Erc20SwapAsset.sol

```
Gas costs:
Gas requirement of function Erc20SwapAsset.changeDCRMOwner 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: 590:4:

Gas costs:
Gas requirement of function Erc20SwapAsset.Swapin 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: 599:4:

Gas costs:
Gas requirement of function Erc20SwapAsset.Swapout 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: 605:4:

Similar variable names:
Erc20SwapAsset.Swapin(bytes32,address,uint256) : Variables have very
similar names "account" and "amount". Note: Modifiers are currently
not considered by this static analysis.
Pos: 600:23:

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: 229:8:
```

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.

Erc20SwapAsset.sol

```
Found more than One contract per file. 6 contracts found!
Pos: 1:4
Code contains empty blocks
Pos: 67:22
Use Custom Errors instead of require statements
Pos: 9:139
Use Custom Errors instead of require statements
Pos: 9:169
Error message for require is too long: 33 counted / 32 allowed
Pos: 9:193
Use Custom Errors instead of require statements
Pos: 9:264
```

Software analysis result:

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

EtherAuthority

Email: audit@EtherAuthority.io