QuillAudits

Audit Report
August, 2024

For

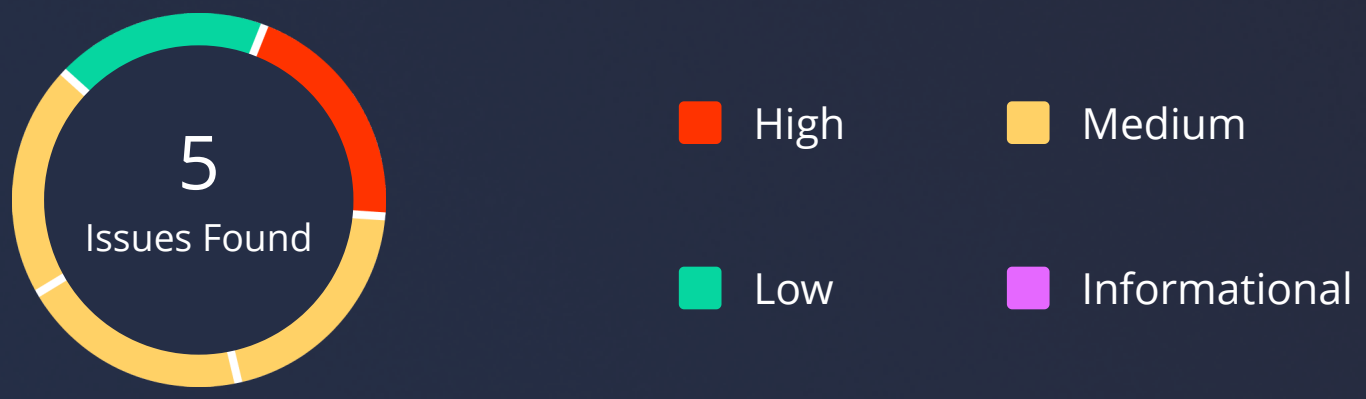
Demex

Table of Content

Executive Summary

Project Name	Demex
Overview	Carbon Bridge employs proxy upgrade pattern to ensure that key contracts are upgradable and compatible with Axelar's gateways and Carbon's Bridge module. Carbon bridge consists of a gateway contract and its auxiliary contracts that facilitate additional functionalities such as token deployment.
Timeline	30th July 2024 - 19th August 2024
Updated Code Received	21st August 2024
Second Review	26th August 2024 - 27th August 2024
Method	The scope of this audit was to analyze the Demex Contracts for quality, security, and correctness.
Audit Scope	The scope of this audit was to analyze the Demex Contracts for quality, security, and correctness.
Source Code	https://github.com/Switcheo/carbon-axelar-evm
Contracts In-Scope	AxelarCarbonExecutable AxelarCarbonGateway CarbonGenericAccessControl CarbonGenericCaller NativeGasTokenDepositor TokenController CarbonWrappedERC20
Branch	Main Commit: 7d07e0fe4e84f509eaa3a75391d5d3ca6d98aba7
Fixed In	https://github.com/Switcheo/carbon-axelar-evm/commit/f269d621f28bb52cd75da05865ebc14db68a9d8e

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	3	1	0

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC's standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Change of name in CarbonWrappedERC20 does not update the DOMAIN_SEPARATOR causing invalid permits.

Path

src/CarbonWrappedERC20.sol

Path

changeName()

Description

This function is used to update the token name, however a serious issue arises when this happens.This contract inherits the solmate ERC20 contract, which initializes the domain separator in the constructor.

One key part of the domain separator being the name , hence when the name is changed, the INITIAL_DOMAIN_SEPARATOR would now be invalid. Hence the permit functionality will be permanently bricked.

POC

```
  /// @notice administrative function to change the name of the token that can be done by ADMIN_GROUP
  function changeName(string memory name_) external onlyRole(DEFAULT_ADMIN_ROLE) {
      name = name_;
  }
  function DOMAIN_SEPARATOR() public view virtual returns (bytes32) {
      return block.chainid == INITIAL_CHAIN_ID ? INITIAL_DOMAIN_SEPARATOR : computeDomainSeparator();
  }

  function computeDomainSeparator() internal view virtual returns (bytes32) {
      return  keccak256(  abi.encode( keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"), keccak256(bytes(name)),
  keccak256("1"),  block.chainid, address(this) )  );  }
```

Recommendation

It is necessary to override the DOMAIN_SEPARATOR() function to always compute the new domain separator. However, there is the issue of this change affecting previous signatures. It is, therefore advised to completely remove the function.

For the first fix, add this to the **CarbonWrappedERC20** contract.

```
function DOMAIN_SEPARATOR() public view override returns (bytes32) {
    return computeDomainSeparator(); }
```

Status

Resolved

Medium Severity Issues

2. function DOMAIN_SEPARATOR() public view override returns (bytes32) {
return computeDomainSeparator(); }

Path

src/CarbonWrappedERC20.sol

Description

The CarbonWrappedERC20 inherits the pause functionality from the CarbonGenericAccessControl, however it has a few issues:

- It does not set the pauser in __token_control_init()
- It does not implement the whenNotPaused and whenPaused modifiers in any of its functions.

Recommendation

Update the __token_control_init() function to set a pauser, also implement the whenNotPaused modifier in the mint, burn, transfer, and transferFrom

Status

Resolved

3. Use onlyInitializing instead of initializer for parent contracts

Path

src/CarbonGenericAccessControl.sol

Function

__control_init

Description

According to the **Open Zeppelin Docs** the initializer should be used in the child contract that has inherited the parent contracts, meanwhile the inherited contracts should use the onlyInitializing. However this is not done here, instead the parent contract and child contract all use the initializer modifier.

From the implementation of all OpenZeppelin Upgradeable contracts being inherited, we see the proper use of modifiers:

```
function __UUPSUpgradeable_init() internal onlyInitializing {
}

function __UUPSUpgradeable_init_unchained() internal onlyInitializing {
}
```

Recommendation

Change the modifier for __control_init to onlyInitializing.

```
function __control_init(address msgSender_) internal onlyInitializing {
    // setup admin role then grant it to deployer
    _grantRole(DEFAULT_ADMIN_ROLE, msgSender_);
```

Status

Resolved

4. Upgradeable contracts are unsafe for upgrades

Path

AxelarCarbonGateway, TokenController, CarbonGenericAccessControl

Description

The AxelarCarbonGateway, TokenController are to be deployed as upgradeable contracts, however this isn't done properly.

For OpenZeppelin contracts used within upgradeable contracts It is required to use the Upgradeable variant of OpenZeppelin Contracts. This is stated in the docs in multiple places:

- Using with Upgrades - OpenZeppelin Docs
- Writing Upgradeable Contracts - OpenZeppelin Docs

Potential issues are the Lack of storage gaps in these library contracts, as well as checks for storage incompatibilities across minor versions, all of which have been added and done by open zeppelin.

Recommendation

Use Open Zeppelin Upgradeable contracts in all contracts that are upgradeable.

Status

Resolved

Low Severity Issues

5. Lack of event for multiple contracts

Path

CarbonGenericCaller, CarbonWrappedERC20, NativeGasTokenDepositor, TokenController

Description

In the contracts listed above, there are multiple events lacking in important functions. It is important to emit events in vital protocol functions.

Recommendation

Add the missing event in:

- CarbonGenericCaller::executeNext()
- CarbonWrappedERC20::changeName()
- NativeGasTokenDepositor::deposit()
- NativeGasTokenDepositor::depositAndExecute()
- TokenController::changeCarbonGenericCaller()

Status

Resolved

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Demex codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, and Low severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.In the End, Demex team resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Demex smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Demex smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Demex to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2024

For

Demex

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com