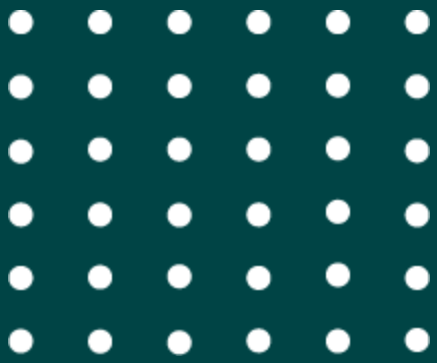
CarbonWrappedGLP



IMMUNEBYTES

17 February 2023

Introduction

About Demex

Demex (Decentralized Mercantile Exchange) is a cross-chain DEX that launched in 2020, with CLOB-based perps, liquidity pools, money market, etc, and is designed to support any type of financial asset imaginable. It is a fully permissionless and non-custodial platform, allowing any user to list new markets, and tokens, and participate in DeFi, giving control back to users for a frictionless trading experience that rivals CEXs.

CarbonWrappedGLP

Carbon's cGLP vault provides the purest and simplest yield for GLP holders by auto-compounding all native ETH yield earned by GLP, with no vault fees or withdrawal fees, so that you earn more with less effort.

More info can be found here:
https://blog.switcheo.com/what-is-cglp-arbitrum-cosmos-first-zero-fee-glp-vault-earn-more-yield-on-demex/

Visit https://dem.exchange/ to learn more about the services.

About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The Demex team has provided the following doc for audit:

1. https://blog.switcheo.com/what-is-cglp-arbitrum-cosmos-first-zero-fee-glp-vault-earn-more-yield-on-demex/
2. https://www.notion.so/immunebytes/Standard-Contract-Dem-Exchange

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

Project Name	Demex
Contract Name	CarbonWrappedGLP.sol
Platform	Arbitrum
Languages	Solidity
GitHub Link	https://github.com/Switcheo/carbon-wrapped-glp
Commit - Initial Audit	https://arbiscan.io/token
Commit - Final Audit	bf6bc5364b8050a8bbd98aca62a57d1dcd62bcd7
Platforms & Tools	Remix IDE, Truffle, VScode, Contract Library, Slither, SmartCheck, Fuzz

Security Level References

Every issue in this report was assigned a severity level from the following:

CRITICAL

Issues may result in fund leakage or incorrect fund allocation.

HIGH

Issues affecting the business logic, performance, and functionality.

MEDIUM

Issues could lead to data loss or other manipulations.

LOW

Issues around smart contract code upgradability, libraries, and others.

INFORMATIONAL

Issues which can further improve the code on gas optimizations and reusability.

Issues	Critical	High	Medium	Low	Informational
Open	-	-	-	-	-
Closed	-	-	1	3	1
Acknowledged	-	1	-	-	-

Findings

#	Findings	Risk	Status
1.	Assets can be withdrawn even if the contract is Paused	**High**	**Acknowledged**
2.	.call() should be preferred instead of transfer()	**Medium**	**Fixed**
3.	Return Value of an External Call is not used effectively	**Low**	**Fixed**
4.	Absence of Zero Address Validation in emergencyRetrieve() function	**Low**	**Fixed**
5.	Constant declaration should be preferred	**Low**	**Fixed**
6.	Coding Style Issues in the Contract	**Informational**	**Fixed**

Critical Severity Issues

No issues were found.

High Severity Issues

1. **Assets can be withdrawn even if the contract is Paused**
   Line no: 78

   Description:
   As per the current architecture prohibits any deposit or withdrawal of assets in scenarios where the contract has been paused.
   This is done by attaching a whenNotPaused() modifier to functions like deposit() or redeem().

   However, during the manual review of the contract, it was found that no such modifier has been attached to the redeemWithoutCompound() function. It leads to a scenario where redeeming assets is still possible with a paused contract, which doesn't seem to be an intended behavior of the contract considering the overall architecture.

   Recommendation:
   If the above-mentioned design is not intentional, it is highly recommended to attach imperative modifiers or validations to important functions.

   Acknowledged (February 17th, 2023): The team has acknowledged the issue.

   Comment from the team:
   This is intentional, to allow users to withdraw their funds when contract is paused.

Medium severity issues

1. **.call() should be preferred instead of transfer()**
   Line no: 114

   Description:
   The emergencyRetrieve() function allows the admin to withdraw native tokens from the contract to any specific user or contract address, called the recipient address.

   The function uses the .transfer() method to initiate a transfer of ether from the contract to the address.

   ```
   function emergencyRetrieve(address tokenAddress, address payable recipient, uint256 amount) external onlyOwner {
     // disable vault asset withdraw
     require(tokenAddress != address(sGLP), "CarbonWrappedGLP: cannot emergency retrieve asset");

     if (tokenAddress == address(0)) {
       recipient.transfer(address(this).balance); // Uses transfer()
     }
   ```

   However, if the recipient of the ether is a smart contract, the transfer might fail. This is because methods like transfer() & send() when used for sending ether, always forward a fixed amount of gas, i.e., 2300.

   If the recipient smart contract uses more than this amount of gas to execute any further transaction, the transfer of ether will never succeed in such a scenario.

   Recommendation:
   It is recommended to use the call() method for initiating ether transfers from the contract instead of transfer() or send(). For more details, refer to this article.

   Amended (February 17th, 2023): The team has fixed the issue, and it is no longer present in commit bf6bc5364b8050a8bbd98aca62a57d1dcd62bcd7.

Low severity issues

1. **Return Value of an External Call is not used effectively**
   Line no: 166

   Description:
   The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.

   These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.
   However, the contract never uses these return values throughout the contract.

   Recommendation:
   Effective use of all the return values from external calls must be ensured within the contract.

   Amended (February 17th, 2023): The team has fixed the issue, and it is no longer present in commit bf6bc5364b8050a8bbd98aca62a57d1dcd62bcd7.

2. **Absence of Zero Address Validation in emergencyRetrieve() function**
   Line no: 161-176

   Description:
   The emergencyRetrieve() function in the contract allows the transfer of native tokens to a specific recipient address.

   However, during the automated testing of the contact it was found that no Zero Address Validation is included to ensure that the address argument being passed is valid.

   Recommendation:
   A require statement should be included in such functions to mitigate any chances of invalid arguments being passed.

   Amended (February 17th, 2023): The team has fixed the issue, and it is no longer present in commit bf6bc5364b8050a8bbd98aca62a57d1dcd62bcd7.

3. **Constant declaration should be preferred**
   Line no: 18

   Description:
   State variables that are not supposed to change throughout the contract should be declared as constant.

   Recommendation:
   The following state variable can be declared as constant unless the current contract design is intended.
   - GLP_MANAGER_ADDRESS

   Amended (February 17th, 2023): The team has fixed the issue, and it is no longer present in commit bf6bc5364b8050a8bbd98aca62a57d1dcd62bcd7.

Informational

1. **Coding Style Issues in the Contract**

   Description:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

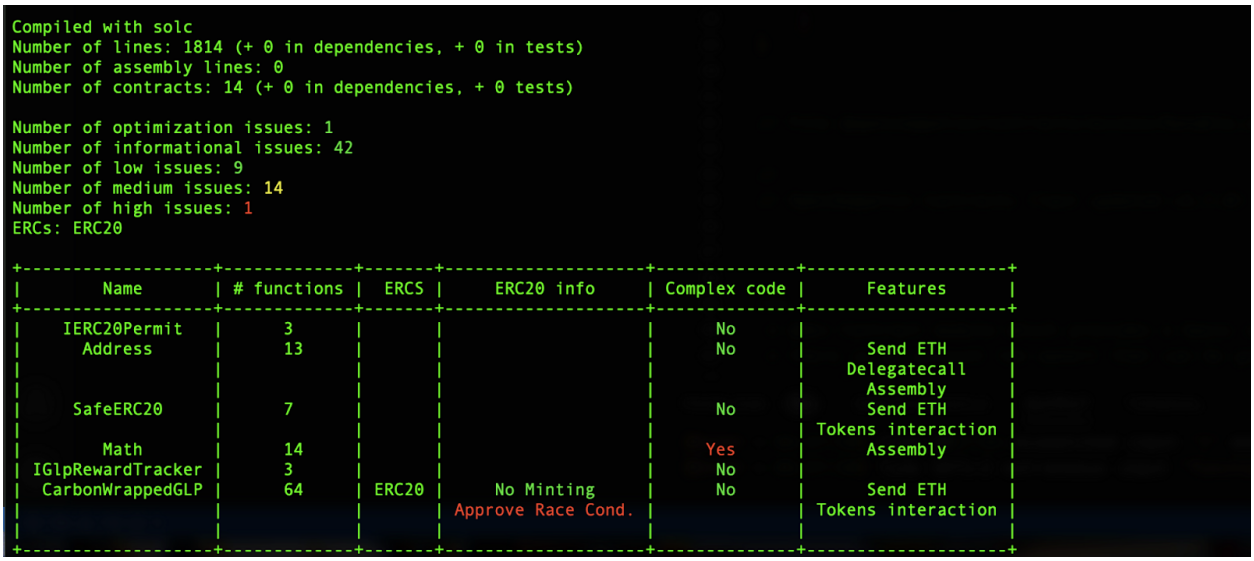
```
Variable CarbonWrappedGLP.GLP_MANAGER_ADDRESS (flat.sol#1655) is not in mixedCase
Constant CarbonWrappedGLP.glpRewardTracker (flat.sol#1657) is not in UPPER_CASE_WITH_UNDERSCORES
Constant CarbonWrappedGLP.glpStakeRouter (flat.sol#1658) is not in UPPER_CASE_WITH_UNDERSCORES
Constant CarbonWrappedGLP.wETH (flat.sol#1659) is not in UPPER_CASE_WITH_UNDERSCORES
Constant CarbonWrappedGLP.fsGLP (flat.sol#1660) is not in UPPER_CASE_WITH_UNDERSCORES
Constant CarbonWrappedGLP.sGLP (flat.sol#1661) is not in UPPER_CASE_WITH_UNDERSCORES
```

   During the automated testing, it was found that the CarbonWrappedGLP contract had quite a few code-style issues. Please follow this link to find details on naming conventions in solidity code.

   Recommendation:
   Therefore, it is recommended to fix issues like naming convention, indentation, and code layout issues in a smart contract.

   Amended (February 17th, 2023): The team has fixed the issue, and it is no longer present in commit bf6bc5364b8050a8bbd98aca62a57d1dcd62bcd7.

Automated Audit Result



```
Compiled with solc
Number of lines: 1814 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 14 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 1
Number of informational issues: 42
Number of low issues: 9
Number of medium issues: 14
Number of high issues: 1
ERCs: ERC20

+--------------------+-------------+-------+--------------------+--------------+--------------------+
|        Name        | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
|    IERC20Permit    |      3      |       |                    |      No      |                    |
|      Address       |      13     |       |                    |      No      |      Send ETH      |
|                    |             |       |                    |              |    Delegatecall    |
|                    |             |       |                    |              |      Assembly      |
|     SafeERC20      |      7      |       |                    |      No      |      Send ETH      |
|                    |             |       |                    |              | Tokens interaction |
|        Math        |      14     |       |                    |     Yes      |      Assembly      |
| IGlpRewardTracker  |      3      |       |                    |      No      |                    |
|  CarbonWrappedGLP  |      64     | ERC20 |     No Minting     |      No      |      Send ETH      |
|                    |             |       | Approve Race Cond. |              | Tokens interaction |
|                    |             |       |                    |              |                    |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
```

Concluding Remarks

While conducting the audits of the Demex smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues along with a few recommendations.

Our auditors suggest that High, Medium, and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Note: The team has already deployed the contract on the mainnet prior to the audit and has acknowledged/fixed the issues locally. The team will redeploy the contract later with improvements.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore running a bug bounty program to complement this audit is strongly recommended.

Our team does not endorse the Demex platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

IMMUNEBYTES

AUDITS