DeHacker

Code Security Assessment

MAPOmnichain Mos

April 4th, 2024

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.
Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

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

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	ButterSwap
Platform	Map Protocol
website	https://www.butterswap.io/swap
Type	Dex
Language	Solidity
Codebase	https://github.com/butternetwork/butter-router-contracts

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	0	0	1
Medium	1	0	0	0	0	1
Minor	0	0	0	0	0	0
Informational	1	0	0	0	0	1
Low	1	0	0	0	0	1

Audit scope

ID	File	SHA256 Checksum
MNS	MAPOmnichainServiceRelayV2.sol	ba0100f849eee6b4a0ee29314225260f3bbed5f45d0f13ab576acdfe9cbbdb8f
VTL	VaultTokenV2.sol	a0599dc88232f3660ef211ef3daad7c8a0d1c69ce356cafbb166f5bdd50bd1ba
TRS	TokenRegisterV2.sol	2c28cb42c9d270d9d3cc955dff9f0d865aded5314b3908668e52e24c3801b00b

Findings

ID	Issue	Severity	Status
MNS-01	Centralization Related Risks	Major	Resolved
MNS-02	Reentrancy Events	Medium	Resolved
TRS-01	Missing Zero Address Validation	Low	Resolved
VTL-01	Missing Events	Informational	Resolved

MAJOR

MNS-01 | Centralization Related Risks

Isuue	Severity	Location	Status
Centralization Related Risks	Major	MAPOmnichainServiceRelayV2.sol: 01	Resolved

Description

The owner of the MAPOmnichainServiceRelayV2 contract has the following permissions:
- function setPause()
- function setUnpause()
- function setTokenRegister()
- function setLightClientManager()
- function registerChain()
- function setDistributeRate()
- function changeAdmin()

The owner of the MAPOmnichainServiceV2 contract has the following permissions:
- function setPause()
- function setUnpause()
- function setLightClient()
- function addMintableToken()
- function removeMintableToken()
- function setRelayContract()
- function registerToken()
- function changeAdmin()

The owner will have the ability to influence the operation results of the protocol.

Recommendation

This finding describes the level of decentralization of the project, and it is recommended to strengthen security and improve the degree of decentralization from the following aspects:
- It is recommended that privileged addresses use multi-signature wallet addresses.

- For modification operations that affect protocol operation stability and key business parameters, it is recommended to implement time locks.

MEDIUM

MNS-02 | Reentrancy Events

Issue	Severity	Location	Status
Reentrancy Events	Medium	MAPOmnichainServiceRelayV2.sol: 02	Resolved

Description

Line 413

```
function _swapOut(
    address _token, // src token
    bytes memory _to,
    address _from,
    uint256 _amount,
    uint256 _toChain, // target chain id
    bytes calldata _swapData
) internal returns (bytes32 orderId) {
    bytes memory toToken = tokenRegister.getToChainToken(_token, _toChain);
    // bytes memory toToken = "0x0";
    require(!Utils.checkBytes(toToken, bytes("")), "Out token not registered");
    orderId = _getOrderId(_from, _to, _toChain);
    (uint256 mapOutAmount, uint256 outAmount) = _collectFee(_token, _amount, selfChainId, _toChain);
    emit CollectFee(orderId, _token, (_amount - mapOutAmount));

    if (tokenRegister.checkMintable(_token)) {
        IMintableToken(_token).burn(mapOutAmount);
    }
    _notifyLightClient(_toChain, bytes(""));
    emit mapSwapOut(selfChainId, _toChain, orderId, toToken, Utils.toBytes(_from), _to, outAmount, _swapData);
}
```

This may cause issues for off-chain components that rely on the values of events.

Recommendation

Confirm that there are no logical issues, or eliminate the reentrant.

LOW

TRS-01 | Missing Zero Address Validation

Issue	Severity	Location	Status
Missing Zero Address Validation	Low	TokenRegisterV2.sol:01	Resolved

Description

Line 277

```
function getVaultBalance(address _token, uint256 _chainId) public view returns (int256 _vaultBalance) {
    address vault = this.getVaultToken(_token);
    (bool result, bytes memory data) = vault.staticcall(abi.encodeWithSignature("vaultBalance(uint256)", _chainId));
    if (result && data.length > 0) {
        _vaultBalance = abi.decode(data, (int256));
        if (_vaultBalance > 0) {
            uint256 tem = this.getToChainAmount(_token, uint256(_vaultBalance), _chainId);
            require(tem <= uint256(type(int256).max), "value doesn't fit in an int256");
            _vaultBalance = int256(tem);
        } else {
            _vaultBalance = 0;
        }
    } else {
        _vaultBalance = 0;
    }
}
```

Non-zero check for the token contract address is missing.

Recommendation

Check if the treasury token contract address is the zero address.

INFORMATIONAL

VTL-01 | Missing Events

Issue	Severity	Location	Status
Uninitialized Local Variables	Informational	VaultTokenV2.sol: 01	Resolved

Description

Line 103

```
function transferToken(
    uint256 _fromChain,
    uint256 _amount,
    uint256 _toChain,
    uint256 _outAmount,
    uint256 _relayChain,
    uint256 _fee
) external override onlyManager {
    vaultBalance[_fromChain] += _amount.toInt256();
    vaultBalance[_toChain] -= _outAmount.toInt256();

    uint256 fee = _amount - _outAmount - _fee;
    vaultBalance[_relayChain] += fee.toInt256();
    totalVault += fee;
}
```

Uninitialized local variables.

Recommendation

Initialize all the variables.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

- Ethereum
- Cosmos
- Eos
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- c++

CONTACTS

- https://dehacker.io
- https://twitter.com/dehackerio
- https://github.com/dehacker/audits_public
- https://t.me/dehackerio
- https://blog.dehacker.io/

DeHacker

April 2024