Cyberscope

Audit Report

MuQuant Vault

June 2023

Vault c64a59996b4d62e3f47eb3e3bbca9d8c0e3fa76f894f93772dff37c1b5b30694

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x9674541F1148C418d4C08FCA888A619764EdE489

Audit Updates

Initial Audit	10 Jun 2023

Source Files

Filename	SHA256
Vault.sol	c64a59996b4d62e3f47eb3e3bbca9d8c0e3fa76f894f93772dff37c1b5b30694

Overview

The Vault contract operates as a vault where some addresses have the authority to deposit and withdraw amount. The deposited tokens are defined dynamically by the authorized addresses during the deposit phase. The `deposit()` and `withdraw()` method names are misleading since the deposit method does not transfer tokens. So, it assumes that the caller is responsible for transferring the amount and calling the `deposit()` method. The applicable users are defined during the deployment phase by the contract owner. The contract owner has the authority to enable and disable the applicable users later.

Roles

- Keeper, has the authority to execute the deposit and withdraw methods.
- Guard, has the authority to enable and disable the keeper addresses.

Findings Breakdown

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

Diagnostics

Severity	Code	Description	Status
●	MEE	Missing Events Emission	Unresolved
●	MU	Modifiers Usage	Unresolved
●	KDO	Keeper Disable Optimization	Unresolved
●	SPI	Stable Parameter Inconsistency	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

MEE - Missing Events Emission

Criticality	Minor / Informative
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

Some methods that are included in the event emmision are:

- deposit
- withdraw
- safe

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	vault.sol#L23
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(keeper[msg.sender], "Vault: not keeper");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

KDO - Keeper Disable Optimization

Criticality	Minor / Informative
Location	Vault.sol#L17
Status	Unresolved

Description

The contract uses 2 mappings to store the vault depositor roles. The guard role has the authority to disable/enable a keeper. Since all the keepers are enabled by default keeping 2 mappings with the same information unnecessarily increases the gas consumption.

```
for (uint256 i = 0; i < keepers.length; i++) {
    keeper[keepers[i]] = true;
    mkeeper[keepers[i]] = true;
}
```

Recommendation

The contract could initially store the keepers to the `keepers` mapping. If the guard disable a keeper, then the `mkeeper` could be updated. For instance, this could be a potential approach:

```
function safe(address _keeper) external {
    require(msg.sender == Guard, "Vault: not guard");
    require(mkeeper[_keeper] || keeper[_keeper], "Vault: not
keeper");
    if (!mkeeper[_keeper]) {
        mkeeper[_keeper] = true;
    }
    keeper[_keeper] = !keeper[_keeper];
}
```

SPI - Stable Parameter Inconsistency

Criticality	Minor / Informative
Location	Vault.sol#L23
Status	Unresolved

Description

According to the contract, only the initial depositor of the stable addresses is registered to a specific token. The next depositor's `stable` parameter is ignored. This may produce wrong assumptions that the amount has been deposited and registered to a specific stable address.

```
function deposit(address token, address stable, uint256 amount)
external {
    require(keeper[msg.sender], "Vault: not keeper");
    if (vaults[token].stable == address(0)) {
        vaults[token].stable = stable;
    }
    vaults[token].amount += amount;
}
```

Recommendation

The team is advised to check if the `stable` parameter matches the initial `stable` address. If it does not, then it could revert with a descriptive event that describes the diversion between the initial `stable` parameter.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Vault.sol#L16
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
Guard
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Vault.sol#L8,43
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address public Guard
address _keeper
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Vault.sol#L16
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
Guard = guard
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Vault.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.9;
```

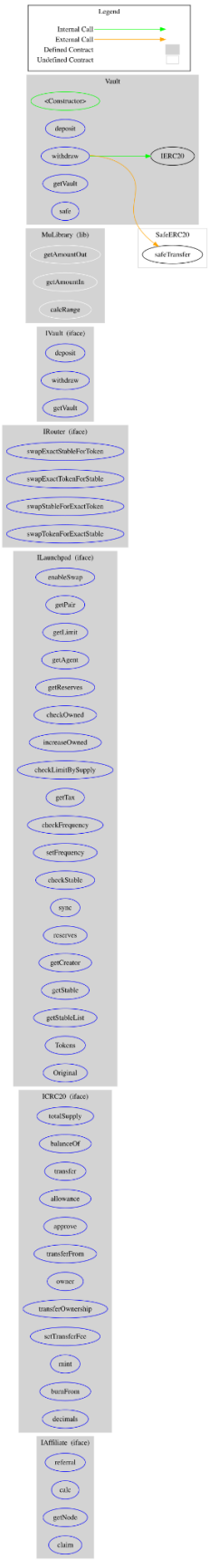
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IVault	Interface			
	deposit	External	✓	-
	withdraw	External	✓	-
	getVault	External		-
Vault	Implementation			
		Public	✓	-
	deposit	External	✓	-
	withdraw	External	✓	-
	getVault	External		-
	safe	External	✓	-

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Vault
<Constructor>
deposit
withdraw
getVault
safe
IERC20
MuLibrary (lib)
getAmountOut
getAmountIn
calcRange
SafeERC20
safeTransfer
IVault (iface)
deposit
withdraw
getVault
IRouter (iface)
swapExactStableForToken
swapExactTokenForStable
swapStableForExactToken
swapTokenForExactStable
ILaunchpad (iface)
enableSwap
getPair
getLimit
getAgent
getReserves
checkOwned
increaseOwned
checkLimitBySupply
getTax
checkFrequency
setFrequency
checkStable
sync
reserves
getCreator
getStable
getStableList
Tokens
Original
ICRC20 (iface)
totalSupply
balunceOf
transfer
allowance
approve
transferFrom
owner
transferOwnership
setTransferFee
mint
burnFrom
decimals
IAffiliate (iface)
referral
calc
getNode
claim

Summary

MuQuant Valut contract implements a vault mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io