VOLMEX CORE SMART CONTRACTS AUDIT

by Aditya Rout

Document version: v1

Stage: Final

Table of contents

Table of contents **2**

Introduction **3**
- Auditing Approach and Methodologies applied 3
- Audit Details 3

Audit Goals **4**
- Security 4
- Sound Architecture 4
- Code Correctness and Quality 4

Issue Categories **5**
- High level severity issues 5
- Medium level severity issues 5
- Low level severity issues 5

Manual Audit **6**
- Low level severity issues 6
- Medium level severity issues 7
- Recommendations 8

Automated Audit **9**
- Remix Compiler Warnings 9
- Contract Library 9
- SmartCheck 10
- Slither 11

Disclaimer **20**

Summary **20**

Introduction

This Audit Report mainly focuses on the overall security of Volmex core Smart Contracts. With this report, I have tried to ensure the reliability and correctness of the smart contracts involved by a complete and rigorous assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

Rigorous testing of the project was performed starting with analyzing the code design patterns in which the smart contract architecture was reviewed to ensure it is structured and had safe usage of third-party smart contracts and libraries.

Then a formal line-by-line inspection of the Smart Contract was performed to find any potential issues like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In Automated Testing, the Smart Contracts were tested using popular automation tools to identify vulnerabilities and security flaws.

The code analysis methodologies included -

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

Audit Details

Project Name: Volmex core
Codebase: https://github.com/volmexfinance/volmex-core/tree/master/contracts
Commit hash: 35617f290b3f4ec8452fe2bf70bb6848463f5fcf
Languages: Solidity (Smart contract), Javascript (test cases)
Platforms and Tools: Remix IDE, Hardhat, Solhint, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient, and working according to the specifications. The audit activities can be grouped into the following three categories:

Security

Identifying security-related issues within each contract and the system of contract.

Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Issue Categories

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

High level severity issues

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	LOW	MEDIUM	HIGH	RECOMMENDATIONS
OPEN	0	0	0	0
CLOSED	3	1	0	2

Manual Audit

For this section, the code was tested/read line by line. Remix IDE's JavaScript VM and Kovan networks were also used to test the contract functionality.

Low level severity issues

- ***Status: Fix planned in next version, closed***

  Description: The following functions are declared public but are not being called from anywhere else in the contract:
  VolmexProtocol.sol:
    - *initialize*

  VolmexPositionToken.sol:
    - *mint*
    - *burn*
    - *pause*
    - *unpause*

  Recommendation: The functions should be declared **external**. This saves gas on function call and contract deployment.

- ***Status: Fix planned in next version, closed***

  Description: In VolmexIndexFactory.sol, the storage variable **positionTokenImplementation** can only be assigned a value upon initialization of the contract. This means the value cannot be updated once set upon deployment of the contract. This is a problem as in case of any typos or mistakes in providing the value initially will mean this value is set as a mistaken value forever and the contract would need to be upgraded or redeployed in order to fix this. This is a costly operation as compared to an update function that could help reset the value if necessary.

  Recommendation: Adding a new function like **updatePositionTokenImplementation** which helps reset the value of **positionTokenImplementation** with the **onlyOwner** modifier attached**.**

- ***Status: Fix planned in next version, closed***

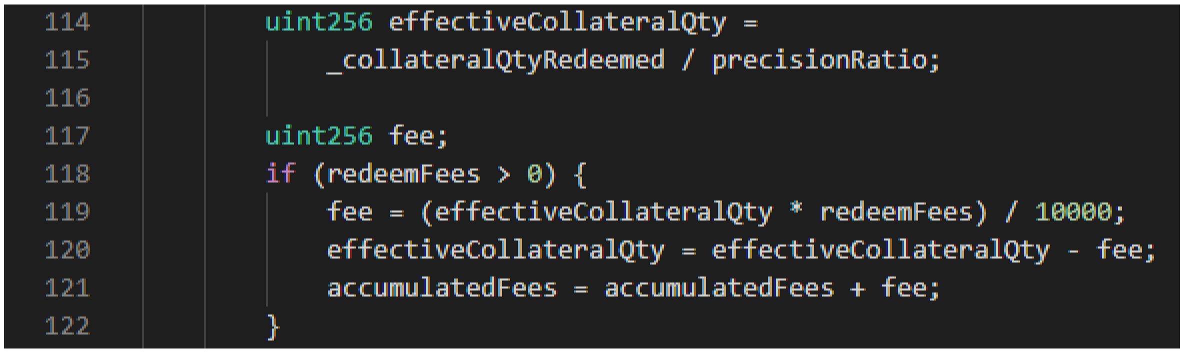
  Description: VolmexProtocolWithPrecision.sol inherits the whole of VolmexProtocol.sol which causes the **collateralize** and **_redeem** functions to be duplicated. As solidity contracts always use the function definition in the parent contract in case of the same definitions, the **collateralize** and **_redeem** functions in VolmexProtocol.sol are never used which causes unnecessary usage of gas on contract deployment.

Recommendation: Removing the inheritance and duplicating the common code of VolmexProtocol.sol in VolmexProtocolWithPrecision.sol. This will make the new VolmexProtocolWithPrecision.sol contract much lighter.

Medium level severity issues

- ***Status: Fixed and closed***
  Commit hash: [23f4e7ca89fd228d79df46124c4aca42eaeb28de]()

  Description: In contract VolmexProtocolWithPrecision.sol, line 119, multiplication is done on the result of a division.



```
114        uint256 effectiveCollateralQty =
115            _collateralQtyRedeemed / precisionRatio;
116
117        uint256 fee;
118        if (redeemFees > 0) {
119            fee = (effectiveCollateralQty * redeemFees) / 10000;
120            effectiveCollateralQty = effectiveCollateralQty - fee;
121            accumulatedFees = accumulatedFees + fee;
122        }
```

effectiveCollateralQty is the division operation between _collateralQtyRedeemed and precisionRatio. In line 119, redeemFees is multiplied with effectiveCollateralQty to calculate the fee which can cause precision issues due to lack of float-based operations in solidity.

Let us consider this example in solidity:

```
// Multiplication on a result of division
uint c = (a / b) * 100000;

// Division on the result of the multiplication
uint d = (a * 100000) / b;

// if a = 523 and b = 10
c = 5200000
d = 5230000
```

As seen in the above example, solidity integer division might truncate. As a result, performing division before multiplication can sometimes cause a loss of precision and value.

Recommendation: Consider ordering multiplication before division. Fee calculation can be modified to be this way:

```
fee = (_collateralQtyRedeemed * redeemFees) / (10000 *
precisionRatio);
```

Recommendations

- ***Status: Fix planned in next version, closed***

  Description: There are uninitialized local variables in the following places:
  - VolmexProtocol.sol, line 213
  - VolmexProtocol.sol, line 370
  - VolmexProtocolWithPrecision.sol, line 87
  - VolmexProtocolWithPrecision.sol, line 117

  This reduces code readability and can cause confusion in implementing some logic using uninitialized variables as it is a unique feature of solidity that uninitialized values are assigned a zero value by default.

  Recommendation: Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability and costs no extra gas.

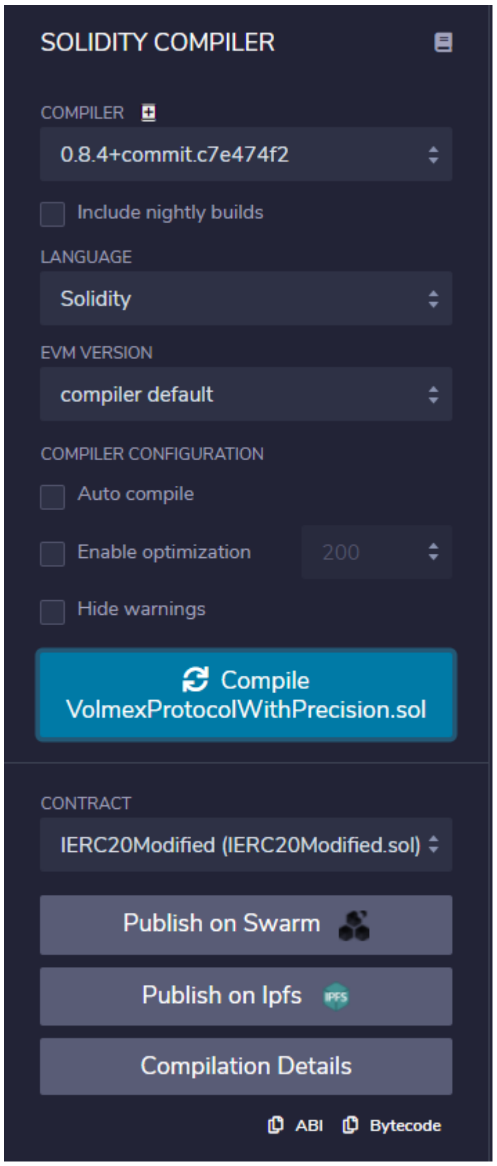
- ***Status: Fix planned in next version, closed***

  Description: Following the solidity naming conventions is recommended. There is a lot of naming in the contracts that deviate from the conventions. For example, *_settlementPrice* does not follow a valid format. As per the docs, it should be either *settlementPrice* or *_settlement_price.*

Automated Audit

Remix Compiler Warnings

All contracts were compiled successfully in remix without any errors or warnings.



SOLIDITY COMPILER

COMPILER: 0.8.4+commit.c7e474f2

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION

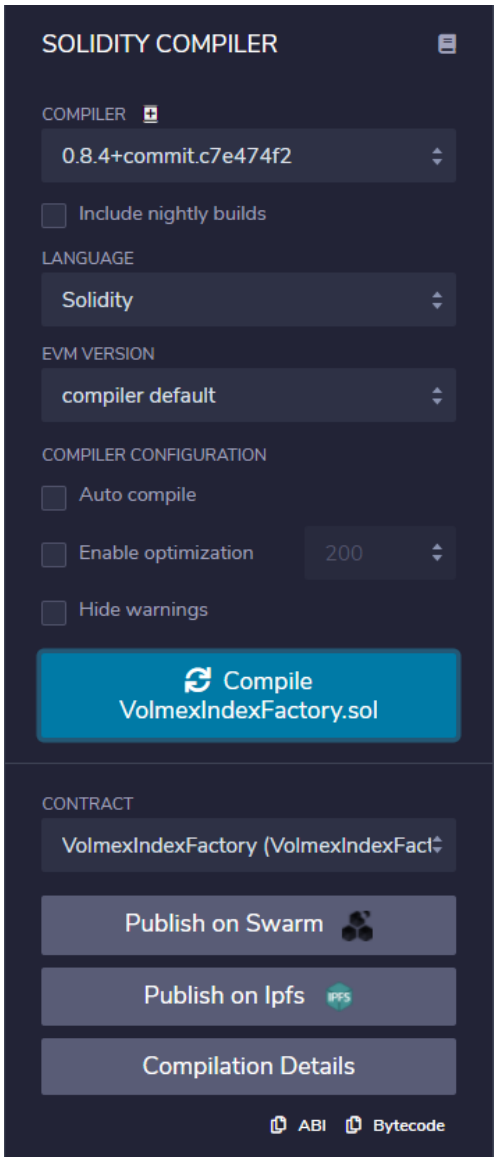
Auto compile

Enable optimization 200

Hide warnings

Compile VolmexProtocolWithPrecision.sol

CONTRACT: IERC20Modified (IERC20Modified.sol)



SOLIDITY COMPILER

COMPILER: 0.8.4+commit.c7e474f2

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

Compile VolmexIndexFactory.sol

CONTRACT: VolmexIndexFactory (VolmexIndexFact

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real-time.

Analysis was performed using the contract Library on the Volmex contracts on kovan:
VolmexIndexFactory.sol:
https://kovan.etherscan.io/address/0x548Fc01d4c398a99672d20A52C391007865e977a

Analysis:
https://contract-library.com/contracts/Kovan/548FC01D4C398A99672D20A52C391007865E977A
VolmexProtocolWithPrecision.sol
https://kovan.etherscan.io/address/0xCE7684c0E0C9B3C1EF3bca33d3B266359e883e34

Analysis:
https://contract-library.com/contracts/Kovan/CE7684C0E0C9B3C1EF3BCA33D3B266359E883E34

It did not return any critical issues during the analysis but a few warnings were raised. One warning to note was related to duplicated functions in VolmexProtocolWithPrecision.sol which has been covered in the manual audit.

SmartCheck

Smartcheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. Smartcheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false-negative rate (FNR).

No major vulnerability was detected upon our analysis. Here is the report:

```
ruleId: SOLIDITY_ADDRESS_HARDCODED
patternId: b140cd
severity: 1
line: 47
column: 50
content: 0x00

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 43
column: 12
content: private
```

```
ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 47
column: 12
content: private

ruleId: SOLIDITY_SHOULD_RETURN_STRUCT
patternId: 83hf3l
severity: 1
line: 93
column: 16
content:
(IERC20ModifiedvolatilityToken,IERC20ModifiedinverseVolatilityToken)

SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA :2
SOLIDITY_ADDRESS_HARDCODED :1
SOLIDITY_SHOULD_RETURN_STRUCT :1
```

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

Upon our analysis, no major vulnerability was detected or has been covered in the manual audit. Here is the report:

```
INFO:Detectors:
OwnableUpgradeable.__gap
(node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable
.sol#74) shadows:
        - ContextUpgradeable.__gap
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#31)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variabl
e-shadowing
```

```
INFO:Detectors:
VolmexProtocolWithPrecision._redeem(uint256,uint256,uint256)
(contracts/protocol/VolmexProtocolWithPrecision.sol#104-139) performs a
multiplication on the result of a division:
        -effectiveCollateralQty = _collateralQtyRedeemed / precisionRatio
(contracts/protocol/VolmexProtocolWithPrecision.sol#114-115)
        -fee = (effectiveCollateralQty * redeemFees) / 10000
(contracts/protocol/VolmexProtocolWithPrecision.sol#119)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before
-multiply
INFO:Detectors:
VolmexProtocol.collateralize(uint256).fee
(contracts/protocol/VolmexProtocol.sol#213) is a local variable never
initialized
VolmexProtocolWithPrecision._redeem(uint256,uint256,uint256).fee
(contracts/protocol/VolmexProtocolWithPrecision.sol#117) is a local
variable never initialized
VolmexProtocol._redeem(uint256,uint256,uint256).fee
(contracts/protocol/VolmexProtocol.sol#370) is a local variable never
initialized
VolmexProtocolWithPrecision.collateralize(uint256).fee
(contracts/protocol/VolmexProtocolWithPrecision.sol#87) is a local variable
never initialized
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized
-local-variables
INFO:Detectors:
Reentrancy in VolmexProtocol.collateralize(uint256)
(contracts/protocol/VolmexProtocol.sol#194-226):
        External calls:
        -
collateral.safeTransferFrom(msg.sender,address(this),_collateralQty)
(contracts/protocol/VolmexProtocol.sol#208)
        State variables written after the call(s):
        - accumulatedFees = accumulatedFees + fee
(contracts/protocol/VolmexProtocol.sol#217)
Reentrancy in VolmexProtocolWithPrecision.collateralize(uint256)
(contracts/protocol/VolmexProtocolWithPrecision.sol#67-102):
        External calls:
        -
collateral.safeTransferFrom(msg.sender,address(this),_collateralQty)
```

```
(contracts/protocol/VolmexProtocolWithPrecision.sol#82)
        State variables written after the call(s):
        - accumulatedFees = accumulatedFees + fee
(contracts/protocol/VolmexProtocolWithPrecision.sol#91)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vu
lnerabilities-2
INFO:Detectors:
Reentrancy in VolmexProtocol._redeem(uint256,uint256,uint256)
(contracts/protocol/VolmexProtocol.sol#365-392):
        External calls:
        - volatilityToken.burn(msg.sender,_volatilityIndexTokenQty)
(contracts/protocol/VolmexProtocol.sol#377)
        -
inverseVolatilityToken.burn(msg.sender,_inverseVolatilityIndexTokenQty)
(contracts/protocol/VolmexProtocol.sol#378-381)
        - collateral.safeTransfer(msg.sender,_collateralQtyRedeemed)
(contracts/protocol/VolmexProtocol.sol#383)
        Event emitted after the call(s):
        -
Redeemed(msg.sender,_collateralQtyRedeemed,_volatilityIndexTokenQty,_invers
eVolatilityIndexTokenQty,fee)
(contracts/protocol/VolmexProtocol.sol#385-391)
Reentrancy in VolmexProtocolWithPrecision._redeem(uint256,uint256,uint256)
(contracts/protocol/VolmexProtocolWithPrecision.sol#104-139):
        External calls:
        - volatilityToken.burn(msg.sender,_volatilityIndexTokenQty)
(contracts/protocol/VolmexProtocolWithPrecision.sol#124)
        -
inverseVolatilityToken.burn(msg.sender,_inverseVolatilityIndexTokenQty)
(contracts/protocol/VolmexProtocolWithPrecision.sol#125-128)
        - collateral.safeTransfer(msg.sender,effectiveCollateralQty)
(contracts/protocol/VolmexProtocolWithPrecision.sol#130)
        Event emitted after the call(s):
        -
Redeemed(msg.sender,effectiveCollateralQty,_volatilityIndexTokenQty,_invers
eVolatilityIndexTokenQty,fee)
(contracts/protocol/VolmexProtocolWithPrecision.sol#132-138)
Reentrancy in VolmexProtocol.claimAccumulatedFees()
(contracts/protocol/VolmexProtocol.sol#339-346):
        External calls:
        - collateral.safeTransfer(owner(),claimedAccumulatedFees)
```

```
(contracts/protocol/VolmexProtocol.sol#343)
        Event emitted after the call(s):
        - ClaimedFees(claimedAccumulatedFees)
(contracts/protocol/VolmexProtocol.sol#345)
Reentrancy in VolmexProtocol.collateralize(uint256)
(contracts/protocol/VolmexProtocol.sol#194-226):
        External calls:
        -
collateral.safeTransferFrom(msg.sender,address(this),_collateralQty)
(contracts/protocol/VolmexProtocol.sol#208)
        - volatilityToken.mint(msg.sender,qtyToBeMinted)
(contracts/protocol/VolmexProtocol.sol#222)
        - inverseVolatilityToken.mint(msg.sender,qtyToBeMinted)
(contracts/protocol/VolmexProtocol.sol#223)
        Event emitted after the call(s):
        - Collateralized(msg.sender,_collateralQty,qtyToBeMinted,fee)
(contracts/protocol/VolmexProtocol.sol#225)
Reentrancy in VolmexProtocolWithPrecision.collateralize(uint256)
(contracts/protocol/VolmexProtocolWithPrecision.sol#67-102):
        External calls:
        -
collateral.safeTransferFrom(msg.sender,address(this),_collateralQty)
(contracts/protocol/VolmexProtocolWithPrecision.sol#82)
        - volatilityToken.mint(msg.sender,qtyToBeMinted)
(contracts/protocol/VolmexProtocolWithPrecision.sol#98)
        - inverseVolatilityToken.mint(msg.sender,qtyToBeMinted)
(contracts/protocol/VolmexProtocolWithPrecision.sol#99)
        Event emitted after the call(s):
        - Collateralized(msg.sender,_collateralQty,qtyToBeMinted,fee)
(contracts/protocol/VolmexProtocolWithPrecision.sol#101)
Reentrancy in VolmexProtocol.togglePause(bool)
(contracts/protocol/VolmexProtocol.sol#353-363):
        External calls:
        - volatilityToken.pause()
(contracts/protocol/VolmexProtocol.sol#355)
        - inverseVolatilityToken.pause()
(contracts/protocol/VolmexProtocol.sol#356)
        - volatilityToken.unpause()
(contracts/protocol/VolmexProtocol.sol#358)
        - inverseVolatilityToken.unpause()
(contracts/protocol/VolmexProtocol.sol#359)
        Event emitted after the call(s):
```

```
        - ToggledVolatilityTokenPause(_isPause)
(contracts/protocol/VolmexProtocol.sol#362)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vu
lnerabilities-3
INFO:Detectors:
VolmexSafeERC20.isContract(address)
(contracts/library/VolmexSafeERC20.sol#107-116) uses assembly
        - INLINE ASM (contracts/library/VolmexSafeERC20.sol#114)
VolmexSafeERC20._verifyCallResult(bool,bytes,string)
(contracts/library/VolmexSafeERC20.sol#118-135) uses assembly
        - INLINE ASM (contracts/library/VolmexSafeERC20.sol#127-130)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usag
e
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['=0.8.4', '^0.8.0']
        - =0.8.4 (contracts/interfaces/IERC20Modified.sol#3)
        - =0.8.4 (contracts/library/VolmexSafeERC20.sol#3)
        - =0.8.4 (contracts/protocol/VolmexProtocol.sol#3)
        - =0.8.4 (contracts/protocol/VolmexProtocolWithPrecision.sol#3)
        - ^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable
.sol#3)
        - ^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable
.sol#4)
        - ^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardU
pgradeable.sol#3)
        - ^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#3)
        - ^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pra
gma-directives-are-used
INFO:Detectors:
ContextUpgradeable.__Context_init()
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#17-19) is never used and should be removed
ContextUpgradeable._msgData()
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#27-30) is never used and should be removed
VolmexProtocol._redeem(uint256,uint256,uint256)
(contracts/protocol/VolmexProtocol.sol#365-392) is never used and should be
removed
VolmexSafeERC20.safeApprove(IERC20,address,uint256)
(contracts/library/VolmexSafeERC20.sol#34-43) is never used and should be
removed
VolmexSafeERC20.safeDecreaseAllowance(IERC20,address,uint256)
(contracts/library/VolmexSafeERC20.sol#50-57) is never used and should be
removed
VolmexSafeERC20.safeIncreaseAllowance(IERC20,address,uint256)
(contracts/library/VolmexSafeERC20.sol#45-48) is never used and should be
removed
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version=0.8.4 (contracts/interfaces/IERC20Modified.sol#3)
necessitates a version too recent to be trusted. Consider deploying with
0.6.12/0.7.6
Pragma version=0.8.4 (contracts/library/VolmexSafeERC20.sol#3) necessitates
a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version=0.8.4 (contracts/protocol/VolmexProtocol.sol#3) necessitates
a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version=0.8.4 (contracts/protocol/VolmexProtocolWithPrecision.sol#3)
necessitates a version too recent to be trusted. Consider deploying with
0.6.12/0.7.6
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable
.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable
.sol#4) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardU
pgradeable.sol#3) necessitates a version too recent to be trusted. Consider
deploying with 0.6.12/0.7.6
Pragma version^0.8.0
```

```
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0
(node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
necessitates a version too recent to be trusted. Consider deploying with
0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-ver
sions-of-solidity
INFO:Detectors:
Low level call in VolmexSafeERC20.functionCall(address,bytes,string)
(contracts/library/VolmexSafeERC20.sol#82-88):
        - (success,returndata) = target.call{value: 0}(data)
(contracts/library/VolmexSafeERC20.sol#86)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-cal
ls
INFO:Detectors:
Parameter
VolmexProtocol.initialize(IERC20Modified,IERC20Modified,IERC20Modified,uint
256,uint256)._collateralTokenAddress
(contracts/protocol/VolmexProtocol.sol#122) is not in mixedCase
Parameter
VolmexProtocol.initialize(IERC20Modified,IERC20Modified,IERC20Modified,uint
256,uint256)._volatilityToken (contracts/protocol/VolmexProtocol.sol#123)
is not in mixedCase
Parameter
VolmexProtocol.initialize(IERC20Modified,IERC20Modified,IERC20Modified,uint
256,uint256)._inverseVolatilityToken
(contracts/protocol/VolmexProtocol.sol#124) is not in mixedCase
Parameter
VolmexProtocol.initialize(IERC20Modified,IERC20Modified,IERC20Modified,uint
256,uint256)._minimumCollateralQty
(contracts/protocol/VolmexProtocol.sol#125) is not in mixedCase
Parameter
VolmexProtocol.initialize(IERC20Modified,IERC20Modified,IERC20Modified,uint
256,uint256)._volatilityCapRatio
(contracts/protocol/VolmexProtocol.sol#126) is not in mixedCase
Parameter VolmexProtocol.updateMinimumCollQty(uint256)._newMinimumCollQty
(contracts/protocol/VolmexProtocol.sol#156) is not in mixedCase
```

```
Parameter VolmexProtocol.updateVolatilityToken(address,bool)._positionToken
(contracts/protocol/VolmexProtocol.sol#175) is not in mixedCase
Parameter
VolmexProtocol.updateVolatilityToken(address,bool)._isVolatilityIndexToken
(contracts/protocol/VolmexProtocol.sol#176) is not in mixedCase
Parameter VolmexProtocol.collateralize(uint256)._collateralQty
(contracts/protocol/VolmexProtocol.sol#194) is not in mixedCase
Parameter VolmexProtocol.redeem(uint256)._positionTokenQty
(contracts/protocol/VolmexProtocol.sol#238) is not in mixedCase
Parameter
VolmexProtocol.redeemSettled(uint256,uint256)._volatilityIndexTokenQty
(contracts/protocol/VolmexProtocol.sol#262) is not in mixedCase
Parameter
VolmexProtocol.redeemSettled(uint256,uint256)._inverseVolatilityIndexTokenQ
ty (contracts/protocol/VolmexProtocol.sol#263) is not in mixedCase
Parameter VolmexProtocol.settle(uint256)._settlementPrice
(contracts/protocol/VolmexProtocol.sol#284) is not in mixedCase
Parameter VolmexProtocol.recoverTokens(address,address,uint256)._token
(contracts/protocol/VolmexProtocol.sol#303) is not in mixedCase
Parameter VolmexProtocol.recoverTokens(address,address,uint256)._toWhom
(contracts/protocol/VolmexProtocol.sol#304) is not in mixedCase
Parameter VolmexProtocol.recoverTokens(address,address,uint256)._howMuch
(contracts/protocol/VolmexProtocol.sol#305) is not in mixedCase
Parameter VolmexProtocol.updateFees(uint256,uint256)._issuanceFees
(contracts/protocol/VolmexProtocol.sol#320) is not in mixedCase
Parameter VolmexProtocol.updateFees(uint256,uint256)._redeemFees
(contracts/protocol/VolmexProtocol.sol#320) is not in mixedCase
Parameter VolmexProtocol.togglePause(bool)._isPause
(contracts/protocol/VolmexProtocol.sol#353) is not in mixedCase
Parameter
VolmexProtocolWithPrecision.initializePrecision(IERC20Modified,IERC20Modifi
ed,IERC20Modified,uint256,uint256,uint256)._collateralTokenAddress
(contracts/protocol/VolmexProtocolWithPrecision.sol#37) is not in mixedCase
Parameter
VolmexProtocolWithPrecision.initializePrecision(IERC20Modified,IERC20Modifi
ed,IERC20Modified,uint256,uint256,uint256)._volatilityToken
(contracts/protocol/VolmexProtocolWithPrecision.sol#38) is not in mixedCase
Parameter
VolmexProtocolWithPrecision.initializePrecision(IERC20Modified,IERC20Modifi
ed,IERC20Modified,uint256,uint256,uint256)._inverseVolatilityToken
(contracts/protocol/VolmexProtocolWithPrecision.sol#39) is not in mixedCase
Parameter
```

```
VolmexProtocolWithPrecision.initializePrecision(IERC20Modified,IERC20Modifi
ed,IERC20Modified,uint256,uint256,uint256)._minimumCollateralQty
(contracts/protocol/VolmexProtocolWithPrecision.sol#40) is not in mixedCase
Parameter
VolmexProtocolWithPrecision.initializePrecision(IERC20Modified,IERC20Modifi
ed,IERC20Modified,uint256,uint256,uint256)._volatilityCapRatio
(contracts/protocol/VolmexProtocolWithPrecision.sol#41) is not in mixedCase
Parameter
VolmexProtocolWithPrecision.initializePrecision(IERC20Modified,IERC20Modifi
ed,IERC20Modified,uint256,uint256,uint256)._ratio
(contracts/protocol/VolmexProtocolWithPrecision.sol#42) is not in mixedCase
Parameter VolmexProtocolWithPrecision.collateralize(uint256)._collateralQty
(contracts/protocol/VolmexProtocolWithPrecision.sol#67) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init()
(node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable
.sol#27-30) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained()
(node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable
.sol#32-36) is not in mixedCase
Variable OwnableUpgradeable.__gap
(node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable
.sol#74) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init()
(node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardU
pgradeable.sol#39-41) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained()
(node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardU
pgradeable.sol#43-45) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap
(node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardU
pgradeable.sol#67) is not in mixedCase
Function ContextUpgradeable.__Context_init()
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#17-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained()
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap
(node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.
sol#31) is not in mixedCase
```

Disclaimer

This audit is not a security warranty, investment advice, or an endorsement of the Volmex core contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

Altogether, the code is well written and demonstrates effective use of abstraction, separation of concerns, and modularity. There were no high severity issues found which means the **contracts are good to be deployed on public EVM chains**. The medium-level issue has been fixed by the team in commit 23f4e7ca89fd228d79df46124c4aca42eaeb28de. The low-level issues and recommendations have been agreed upon by the team and will be fixed in the next version.