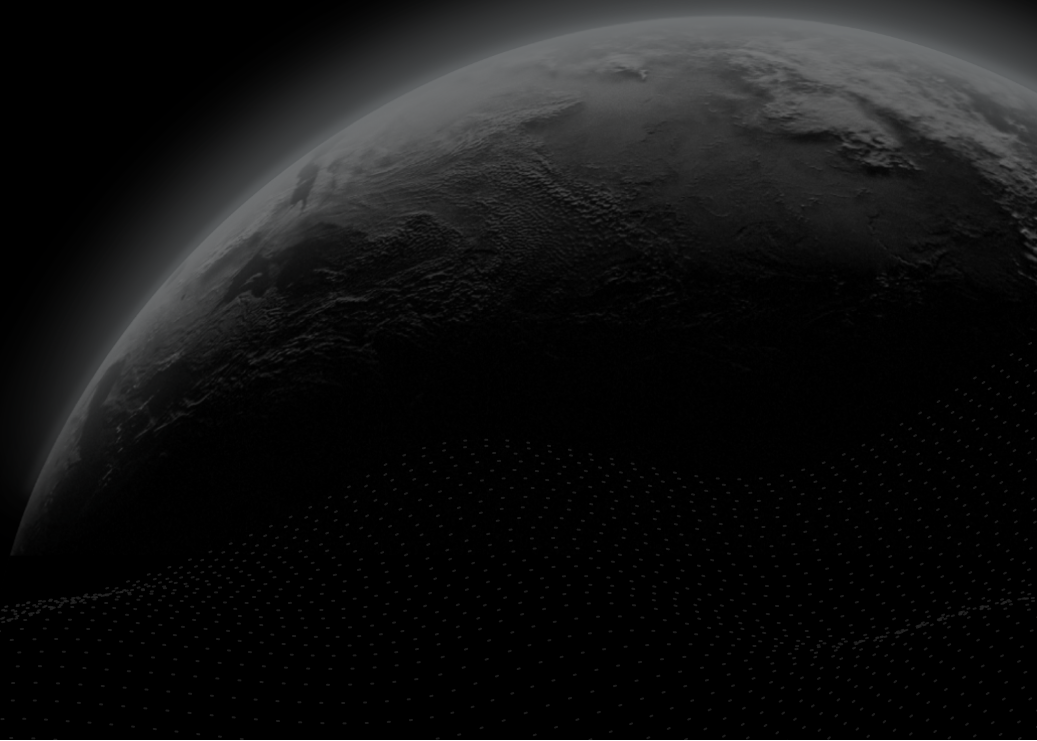
CERTIK

Security Assessment

BLEX

CertiK Assessed on Jul 28th, 2023



CertiK Assessed on Jul 28th, 2023

BLEX

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

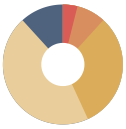
TYPES	ECOSYSTEM	METHODS
Perpetual contracts	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 07/28/2023	Perpetual contracts

CODEBASE
d4bfed4a01816ff5719479283e785aae27c51997
View All in Codebase Page

COMMITS
d4bfed4a01816ff5719479283e785aae27c51997
View All in Codebase Page

Vulnerability Summary



51	28	0	0	23	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

2	Critical	1 Resolved, 1 Acknowledged	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
4	Major	2 Resolved, 2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
16	Medium	10 Resolved, 6 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
23	Minor	14 Resolved, 9 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
6	Informational	1 Resolved, 5 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BLEX

Optimizations

Formal Verification

CODEBASE | BLEX

Repository

d4bfed4a01816ff5719479283e785aae27c51997

Commit

d4bfed4a01816ff5719479283e785aae27c51997

AUDIT SCOPE | BLEX

36 files audited ● 17 files with Acknowledged findings ● 5 files with Resolved findings ● 14 files without findings

ID	Repo	File	SHA256 Checksum
CVB	dalveytech/blex-contract	contracts/vault/CoreVault.sol	b961e1aa74c9b9bd57bf2e496fff675a6991bfa43a660af5cd5aef6c5ca7ab86
RDB	dalveytech/blex-contract	contracts/vault/RewardDistributor.sol	ba945c800498af798ccdd6e3f2d230dc091cb74efcf5fcf8edadc6a474c116dd
VRB	dalveytech/blex-contract	contracts/vault/VaultReward.sol	6c199adc80fbcb08177388f11ae2c9d94c07b5c7c01a28ca7f29d2cc70a21a9a
VRU	dalveytech/blex-contract	contracts/vault/VaultRouter.sol	19541de9764a81cc6a51250396d9ea851bfe292d50fc30a90fc5da5cedf0e6c0
FRB	dalveytech/blex-contract	contracts/fee/FeeRouter.sol	7b86f37e64867d8380ceac5db439213f9ad1800b13989b1dad1504b3ddade618
FVB	dalveytech/blex-contract	contracts/fee/FeeVault.sol	d91988bbbd3f88e9e89196f40f484626990fc845b3a2f24261b8973829a2d1e0
FFB	dalveytech/blex-contract	contracts/fee/FundFee.sol	b84db6600230983b3f501de2b4cdccb2f579f399fa7240402418b4fe6b9436e3
MAR	dalveytech/blex-contract	contracts/market/Market.sol	3ca3292655e81728bb4ef15656fea989a91c9a31d472e7dbbd634a2c28ac3759
MCS	dalveytech/blex-contract	contracts/market/MarketConfigStruct.sol	39bcb3be103a83926fcb3b180dbef983cc004c59c1f22f942715620c1b0019e5
MRB	dalveytech/blex-contract	contracts/market/MarketReader.sol	b7bbc012ba1193d7559c3d97b0b8c8296a9bf450e2ac868a10703fbe890c9f93
MRU	dalveytech/blex-contract	contracts/market/MarketRouter.sol	b82dc81898a0633a4bbfb5b196e1e8da96c4fb95803ba8eeab2f6b8c16b36bbf
MSB	dalveytech/blex-contract	contracts/market/MarketStorage.sol	68e2cd5d8cc3303926fa7c49cf0c25357eb378e57da1b1c6807b2ba556911359
MVB	dalveytech/blex-contract	contracts/market/MarketValid.sol	68454f5e0c8ca6c03e75fcd31b65cf09e134748a17e27d8958d5d91ba5ba6bcf

ID	Repo	File	SHA256 Checksum
OMB	dalveytech/blex-contract	contracts/market/OrderMgr.sol	0c70b0835cf3841b71648a7731249031ff5f8cd410d13cf44816eea16320b28d
PSM	dalveytech/blex-contract	contracts/market/PositionSubMgr.sol	91270e0d6e41bb2989025e752997ffb9f32e86c131d35350be96f52aa30f6981
CPF	dalveytech/blex-contract	contracts/oracle/ChainPriceFeed.sol	0f2fe0fd718910aaa60c6f5f9de17a092bcac6437d25730a7d6f4ebee65e6325
PRI	dalveytech/blex-contract	contracts/oracle/Price.sol	39a75b808ca922793fca125042fce4a04858a3f1f88f9b0d19f2e0a4421d928a
OBB	dalveytech/blex-contract	contracts/order/OrderBook.sol	24fa18a2accd27bd621843a1334838fdd70c988fdf07ebb8410b39d312a55a03
GVB	dalveytech/blex-contract	contracts/market/GlobalValid.sol	e6b1b5e05b87e53b12a747c86ee4eab49e5638a11e7a6f5abee0c9e33ca28b01
PAM	dalveytech/blex-contract	contracts/market/PositionAddMgr.sol	886de4a0283ca07038ab05cf4fb5cc6b4fc9c1827de7abde3eebf34acede5a6b
FPF	dalveytech/blex-contract	contracts/oracle/FastPriceFeed.sol	656c2ac690674a73af52a3a6ccb16661c58151774855f800e17d64974819581f
PSB	dalveytech/blex-contract	contracts/position/PositionStore.sol	035bd0151db84397ff1ea3d0c186da1e6dcc33cc4f6de0c1da9e55498ab03c01
ACA	dalveytech/blex-contract	contracts/ac/Ac.sol	581f055709b49c500885fd99d3fb17252967424184d177f69b37ab9b7dc2d575
REF	dalveytech/blex-contract	contracts/referral/Referral.sol	c2582bd8e4eafcaa5800db3f497d15d8bab4e3e564ec0a798f79badd768cbfd1
THB	dalveytech/blex-contract	contracts/utils/TransferHelper.sol	95c0cefca128e141ac60f39be4ef75e428ee0095fa9083588390693b60e85d57
EVB	dalveytech/blex-contract	contracts/utils/EnumerableValues.sol	3624e25057fff2c095851dfc57744aea76f32f19f008a738ffe3f6884e98f894
CAL	dalveytech/blex-contract	contracts/utils/Calc.sol	889e7560635d66afa386b1102f0171e82c82cfe48cfa903910b5059acc472847
OLB	dalveytech/blex-contract	contracts/order/OrderLib.sol	fde0883c296e1bb01c597cba104177fe83c4b3bacdfb3cf1ec58977ee3b04fb3

ID	Repo	File	SHA256 Checksum
OSB	dalveytech/blex-contract	contracts/order/OrderStore.sol	2b2c8707157c164521cc55dc7324b90dae574c7692ff83686a42c399f0e7d7ec
OSU	dalveytech/blex-contract	contracts/order/OrderStruct.sol	3aee2891266a6848bdb143c13acc449e280698b858c91393fe9b1981c734fd7f
GDT	dalveytech/blex-contract	contracts/market/GlobalDataTypes.sol	051c4a84844658335f6e1d61046886832ea709e940f4e9e9a72f8e22f2ae73bb
MDT	dalveytech/blex-contract	contracts/market/MarketDataTypes.sol	899a2e090672683f3e1d8d8a7730e09f0dfee20e4ddd4a1f768835ae15c98a3a
MFB	dalveytech/blex-contract	contracts/market/MarketFactory.sol	daa166cd63803535edaf27cd60d3a68872ade9c03af7562d8967b50796f3f1b7
MLB	dalveytech/blex-contract	contracts/market/MarketLib.sol	61904539afb791a9874d49629ebc317e66c66b2c5d49c480cba5983536c90c8a
PBB	dalveytech/blex-contract	contracts/position/PositionBook.sol	a9bdc1cf98e6b99e374eea1062b715cecef5d9b097bd0f01ad7d7d539a533248
PSU	dalveytech/blex-contract	contracts/position/PositionStruct.sol	c94d6cc13397506d01c6c6fb585d22eea84e43b1e2120af2058bb5ab78a8af47

APPROACH & METHODS | BLEX

This report has been prepared for BLEX to discover issues and vulnerabilities in the source code of the BLEX project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

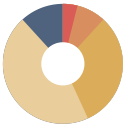
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | BLEX



51	2	4	16	23	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for BLEX. Through this audit, we have uncovered 51 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	**Privileged Accounts Can Move Users' Funds**	**Centralization**	**Critical**	**Acknowledged**
VRU-01	`sell()` Does Not Send Redeemed Tokens To Specified Receiver `to`	Coding Issue	Critical	Resolved
CVB-06	Double Transfers In `_withdraw()`	Coding Issue	Major	Resolved
CVB-07	Potential Frontrunning Attack	Coding Issue	Major	Resolved
GLOBAL-02	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
MAR-05	**Contract Upgrade Centralization Risk**	**Centralization**	**Major**	**Acknowledged**
CON-06	Should Not Collect Negative Fees	Incorrect Calculation	Medium	Resolved
CVB-05	Old Vault Router Is Not Removed From `ROLE_CONTROLLER` When Setting New Vault Router	Coding Issue	Medium	Resolved
FEE-01	Funding Rate Is Always Positive	Design Issue	Medium	Acknowledged
GVB-01	Useable Net Size Calculation Without `isLong` Check	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
MAK-01	Input `isOpen` Is Not Set For `decreasePosition()`	Volatile Code	Medium	Resolved
MVB-04	Not Be Able To `getCollateralRange()`	Incorrect Calculation	Medium	Resolved
MVB-05	Inconsistent Leverage Calculation	Inconsistency	Medium	Resolved
OBB-02	Inconsistent Order Query Condition	Inconsistency	Medium	Resolved
OMB-01	Charge `execFee` Only When The Order Has Sufficient Collateral	Logical Issue	Medium	Resolved
OMB-02	Decrease Collateral Without Checking Liquidation	Logical Issue	Medium	Acknowledged
PAM-01	Check `collateralDelta` Input Before Updating Positions	Volatile Code	Medium	Resolved
PSM-01	Unable To Liquidate Due To Inconsistent Fee Charges	Inconsistency	Medium	Resolved
PSM-06	Unfair Design For Vault Investors	Design Issue	Medium	Acknowledged
PSM-10	Lack Of Trigger Price Validation	Volatile Code	Medium	Acknowledged
VAU-01	Unprotected `initialize()` Function	Coding Issue	Medium	Acknowledged
VRB-01	Reward Calculation May Be Incorrect When Users Call Vault/VaultRouter Buy()/Sell()/Deposit()/Withdraw() Directly	Logical Issue	Medium	Acknowledged
CON-03	Missing Zero Address Validation	Volatile Code	Minor	Resolved
CON-04	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
CON-07	Unsafe Integer Cast	Logical Issue	Minor	Resolved
CPF-02	Inconsistent Price Precision	Inconsistency	Minor	Acknowledged
CVB-08	"VerifyOutAssets" Always Return "True"	Volatile Code	Minor	Resolved
FFB-02	Unused Value	Inconsistency	Minor	Resolved
FRB-02	Inconsistent Fee Value	Inconsistency	Minor	Acknowledged
MAK-02	User Input Oracle Price Will Be Used In The Position Callback If `sizeDelta` Is Zero	Volatile Code	Minor	Resolved
MAK-03	Inconsistent Comments And Validation Logic	Inconsistency	Minor	Resolved
MAK-04	Minimum Slippage Price Is Not Used	Inconsistency	Minor	Acknowledged
MAK-05	Out Of Scope Dependencies	Logical Issue	Minor	Acknowledged
MVB-02	Incorrect Input For `validSize()` Function	Inconsistency	Minor	Resolved
MVB-03	Incorrect Error Message	Inconsistency	Minor	Resolved
OMB-03	Collateral To Be Refunded Should Subtract The `execFee`	Logical Issue	Minor	Resolved
PRI-02	Third-Party Dependency Usage	Volatile Code	Minor	Acknowledged
PSB-02	Lack Of Input Validation On `account` For Function `remove()`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
PSM-02	`transToUser` Should Be Zero If The Collateral Is Insufficient To Cover Fees	Logical Issue	Minor	Resolved
PSM-05	Potentially Insufficient Funds To Take Profits	Volatile Code, Design Issue	Minor	Acknowledged
PSM-07	Lack Of Sanity Check For `liqState`	Volatile Code	Minor	Resolved
PSM-08	Collateral Should Be Decreased To Zero If Collateral Is Insufficient To Cover Fees	Volatile Code	Minor	Resolved
PSM-09	Close The Entire Position Yields A `PartialLiquidation` Reason	Inconsistency	Minor	Resolved
PSM-11	Collects Execution Fee When Removing Orders	Logical Issue	Minor	Acknowledged
VRU-02	Comment About `sell()` Parameter `amount` Does Not Match Actual Code	Coding Issue	Minor	Acknowledged
CVB-01	Potential Incorrect `totalAssets()` Implementation	Coding Issue	Informational	Acknowledged
FFB-03	Typo Of Constant Name	Coding Style	Informational	Resolved
FVB-01	`decreaseFees()` Adds Values On Current Fees	Incorrect Calculation	Informational	Acknowledged
MRB-01	Markets Need To Be Created By The `MarketFactory`	Volatile Code	Informational	Acknowledged
MSB-01	Avoid Potential Storage Conflicts In Market Upgrades	Volatile Code	Informational	Acknowledged
MVB-01	Discussion On Take Profit And Stop Loss Prices	Volatile Code	Informational	Acknowledged

GLOBAL-01 | PRIVILEGED ACCOUNTS CAN MOVE USERS' FUNDS

Category	Severity	Location	Status
Centralization	**Critical**		**Acknowledged**

Description

The `transferOutAssets()` in CoreVault contract, the `withdrawToken()` in RewardDistributor contract, the `transferFromVault()` in VaultRouter contract can be called by privileged accounts to move users' funds out of those contracts.

If those privileged accounts are hacked, all users' funds can be stolen.

Recommendation

It is recommended to remove such a function from the contract. If the project absolutely requires the function, the team should properly secure privileged accounts and document the associated risks in the project documentation. We advise the client to carefully manage the owner account's private key to avoid any potential risks of being hacked.

Alleviation

From BLEX team: "Issue acknowledged. I won't make any changes for the current version."

VRU-01 `sell()` DOES NOT SEND REDEEMED TOKENS TO SPECIFIED RECEIVER `to`

Category	Severity	Location	Status
Coding Issue	Critical	contracts/vault/VaultRouter.sol (v1): 84	Resolved

Description

The `sell()` function is used to sell shares of a vault and send redeemed tokens to user specified address. But the actual code only withdraw tokens from vault to current VaultRouter contract; it does not send redeemed tokens to receiver `to`. It means users can never receive tokens by calling `sell()`. Users' funds are locked in VaultRouter contract.

Recommendation

Recommend reviewing the code and making sure redeemed tokens are sent to user specified address.

Alleviation

Fixed in commit 90e2c8f27bc0f44a20bd82edf02c6ceced5b0845

CVB-06 | DOUBLE TRANSFERS IN `_withdraw()`

Category	Severity	Location	Status
Coding Issue	Major	contracts/vault/CoreVault.sol (v1): 142~148, 151	Resolved

Description

In the `_withdraw()` function, tokens are incorrectly transferred out twice: one is `super._withdraw(...);` ; another one is `SafeERC20.safeTransfer(...);` . Vault router unnecessarily receives user withdrawal amount of tokens. This bug may result in that users can not withdraw the amount of tokens that they own.

Recommendation

It is recommended that the team reviews the code and double-checks if the extra "_withdraw" to the router is the intended design. If it is not, the team should remove or modify the logic of the "_withdraw" implementation accordingly.

Alleviation

Fixed in commit 90e2c8f27bc0f44a20bd82edf02c6ceced5b0845

CVB-07 | POTENTIAL FRONTRUNNING ATTACK

Category	Severity	Location	Status
Coding Issue	Major	contracts/vault/CoreVault.sol (v1): 18	Resolved

Description

"When the vault is empty or nearly empty, deposits are at high risk of being stolen through frontrunning with a "donation" to the vault that inflates the price of a share. This is variously known as a donation or inflation attack and is essentially a problem of slippage." See https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.3/contracts/token/ERC20/extensions/ERC4626.sol

Recommendation

Recommend that vault deployers make an initial deposit of a non-trivial amount of the asset, such that price manipulation becomes infeasible.

Alleviation

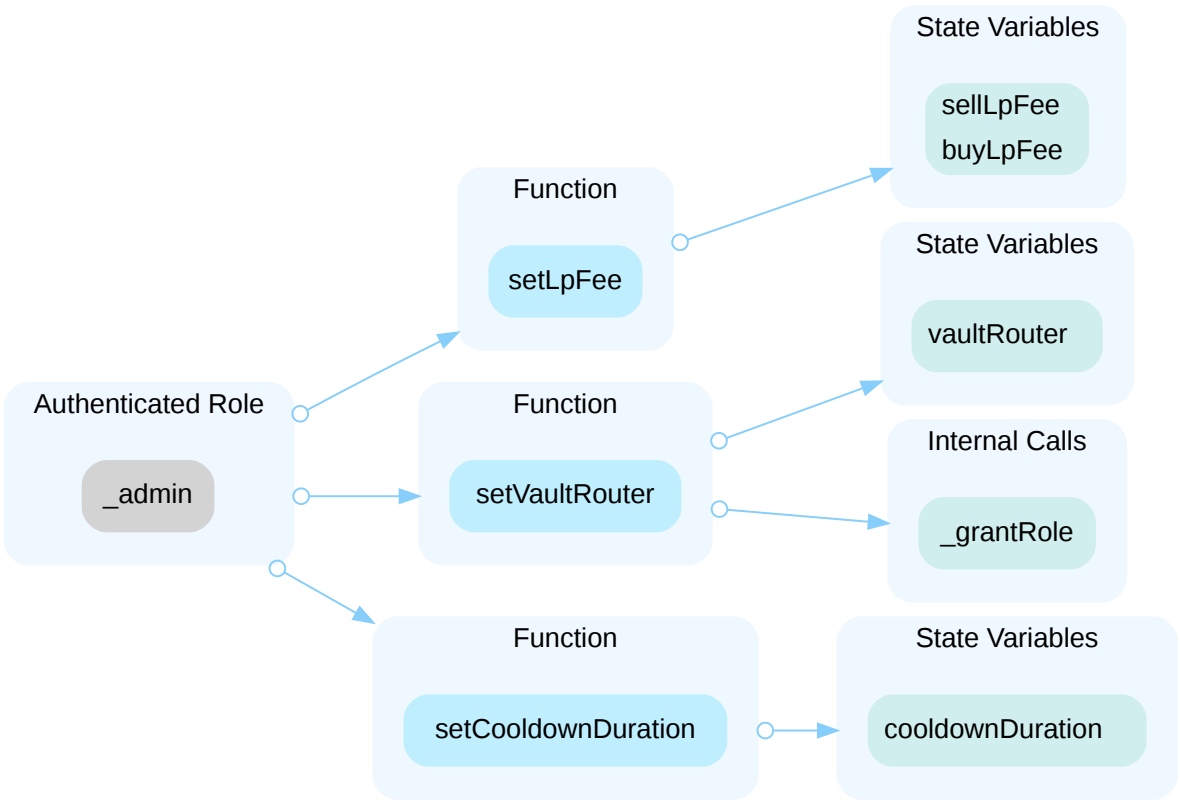
Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

GLOBAL-02 | CENTRALIZATION RELATED RISKS

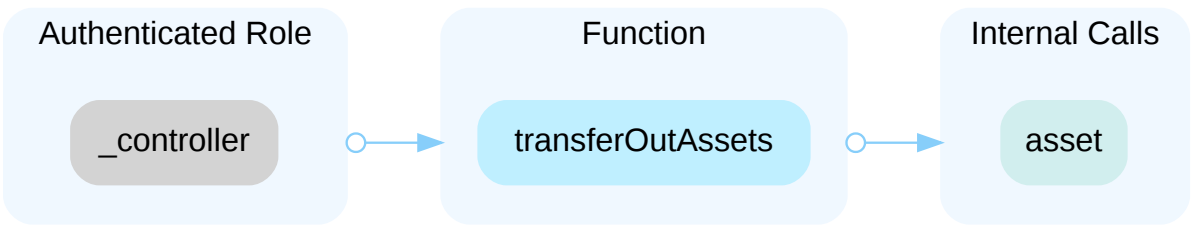
Category	Severity	Location	Status
Centralization	**Major**		**Acknowledged**

Description

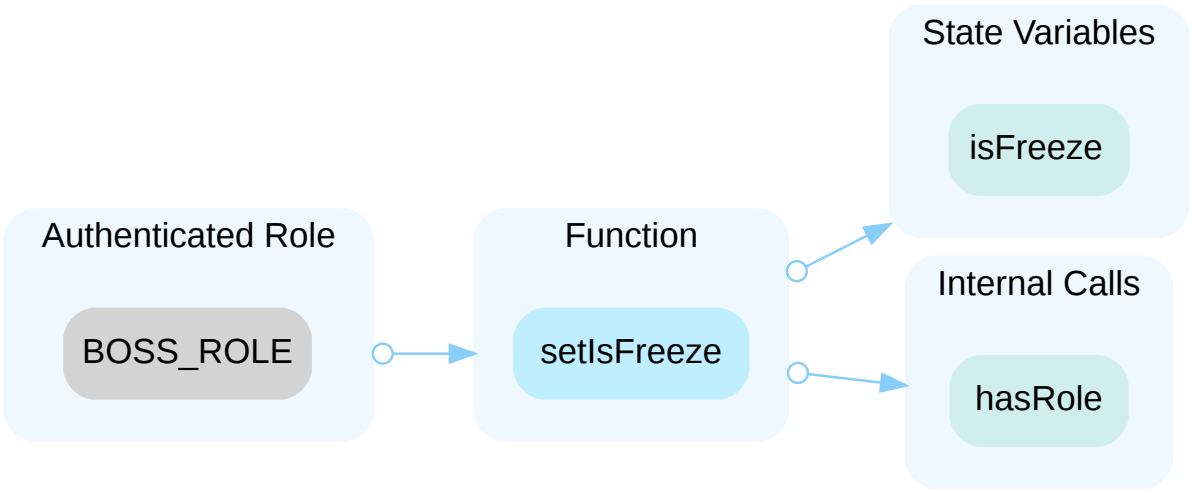
In the contract `CoreVault` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and steal tokens from the contract.



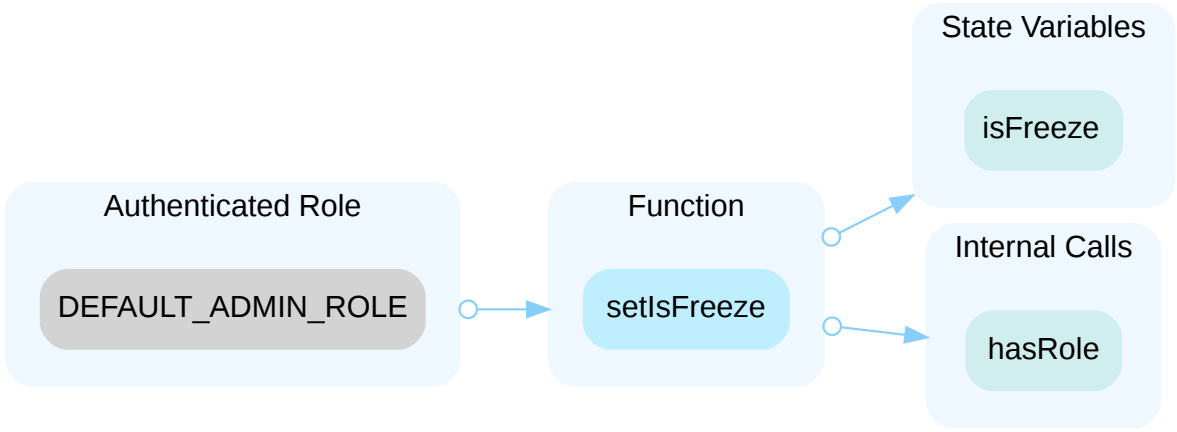
In the contract `CoreVault` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and steal tokens from the contract.



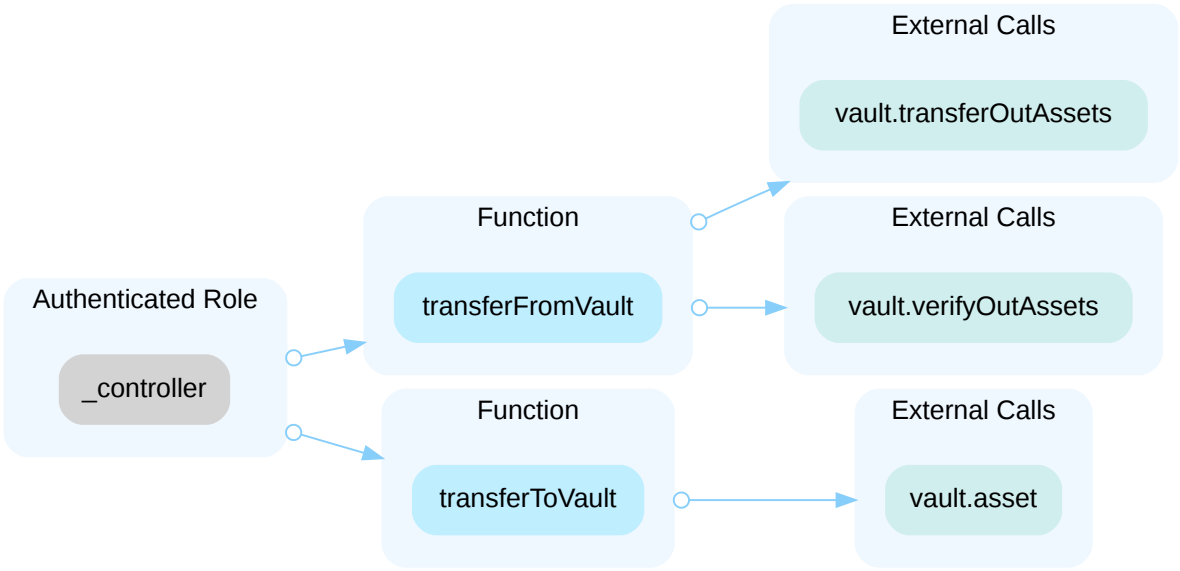
In the contract `VaultRouter` the role `BOSS_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `BOSS_ROLE` account may allow the hacker to take advantage of this authority and make the contract malfunction.



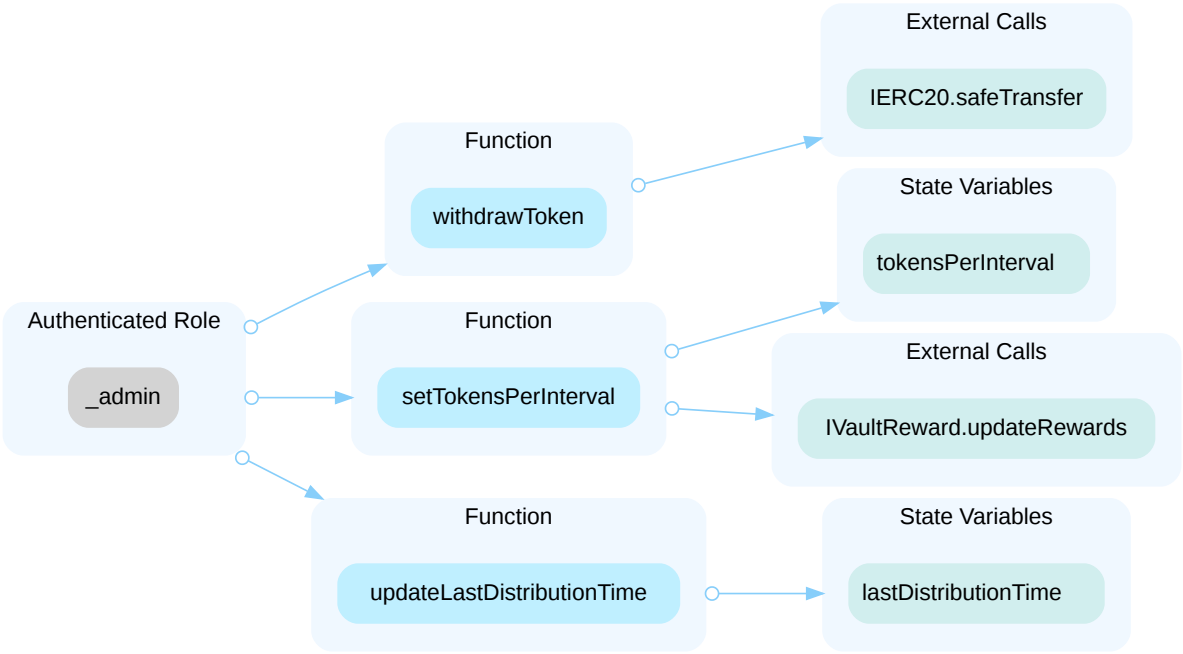
In the contract `VaultRouter` the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and steal tokens from the contract.



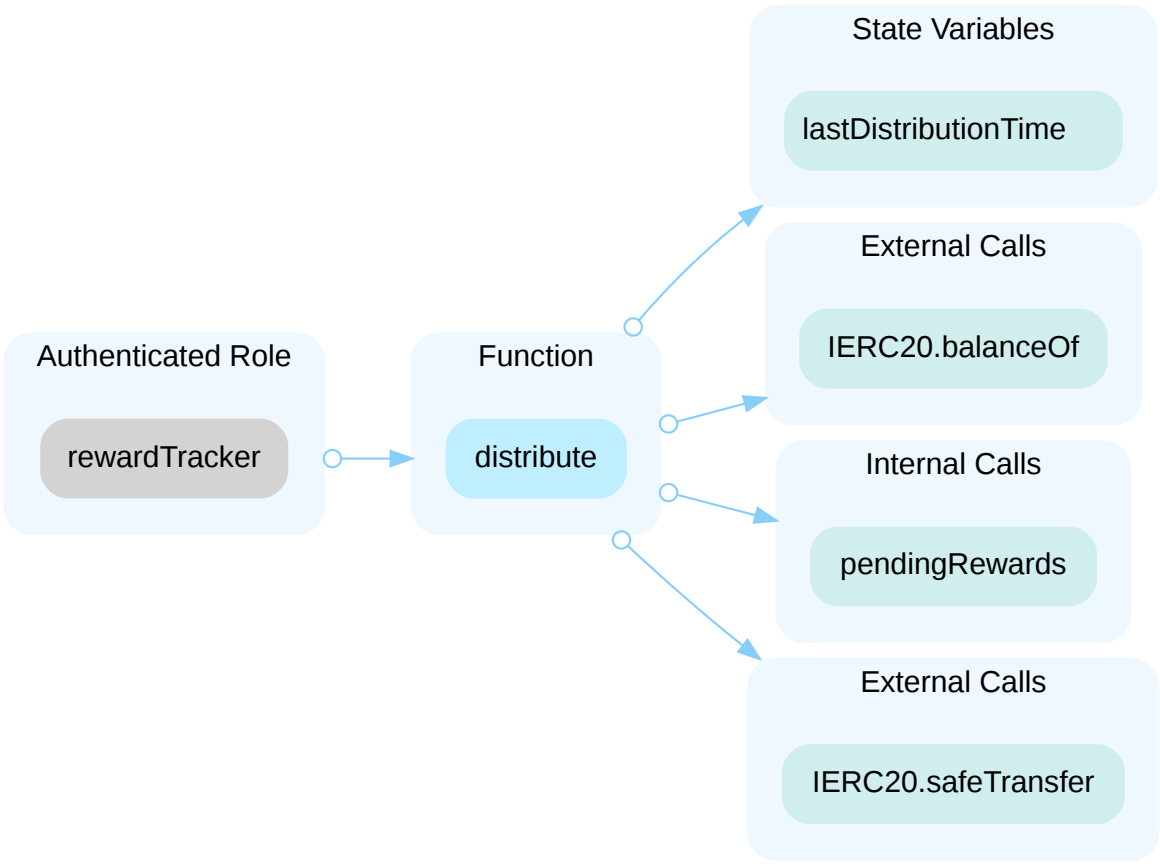
In the contract `VaultRouter` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and steal tokens from the contract.



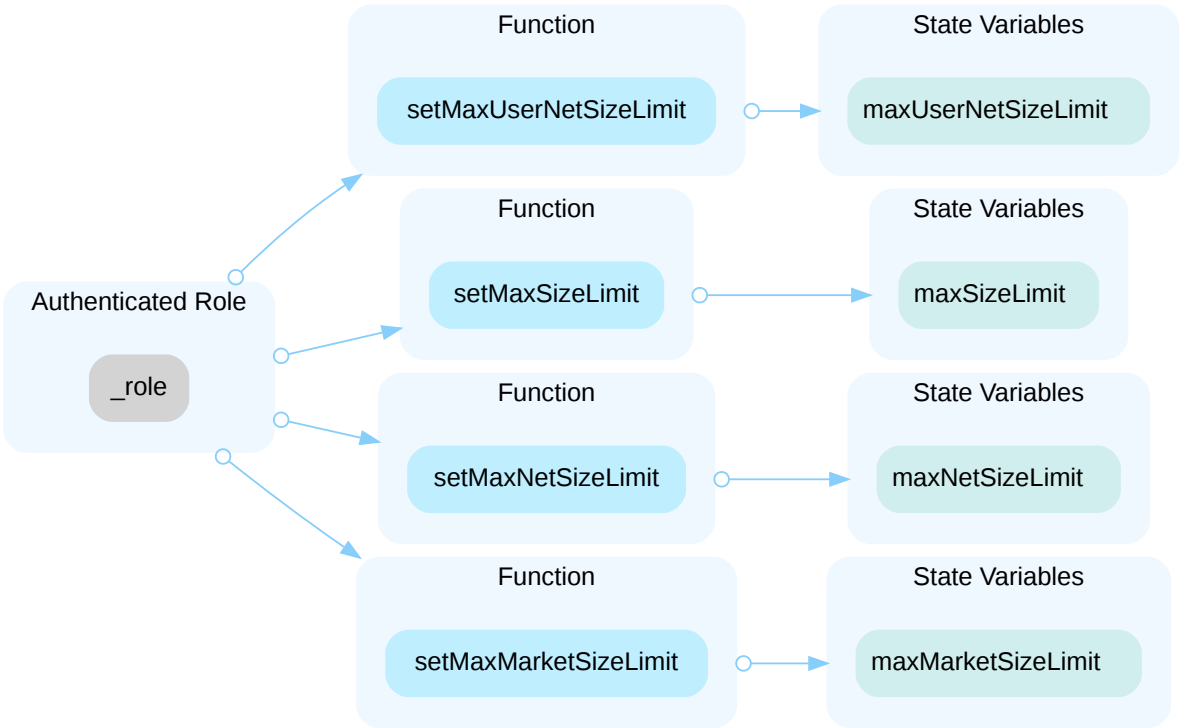
In the contract `RewardDistributor` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and steal tokens from the contract.



In the contract `RewardDistributor` the role `rewardTracker` has authority over the functions shown in the diagram below. Any compromise to the `rewardTracker` account may allow the hacker to take advantage of this authority and steal tokens from the contract.



In the contract `GlobalValid` the role `GLOBAL_MGR_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `_role` account may allow the hacker to take advantage of this authority and set position size limits.



In the contract `Market` the role `MANAGER_ROLE` has authority over the `addPlugin()` function. Any compromise to the `MANAGER_ROLE` account may allow the hacker to take advantage of this authority to add plugin contract addresses.

In the contract `Market` the role `ROLE_CONTROLLER` has authority over the functions below.

- function fallback(): delegate function calls that do not exist in the `Market` contract to `orderMgr` contract.
- function increasePositionWithOrders(): open or increase a position and provide an option to create take-profit or stop-loss orders.
- function decreasePosition(): decrease or close a position.

Any compromise to the `ROLE_CONTROLLER` account may allow the hacker to take advantage of this authority to increase/decrease a position or order.

In the contract `Market` the role `ROLE_POS_KEEPER` has authority over the functions below.

- function liquidatePositions(): liquidate positions.
- function execOrderKey(): execute an order to increase or decrease positions.

Any compromise to the `ROLE_POS_KEEPER` account may allow the hacker to take advantage of this authority to liquidate positions and execute orders.

In the contract `Market` the role `MARKET_MGR_ROLE` has authority over the functions below.

- function setOrderBooks()
- function setPositionBook()
- function setMarketValid()
- function setPositionMgr()
- function setOrderMgr()
- function setPriceFeed()

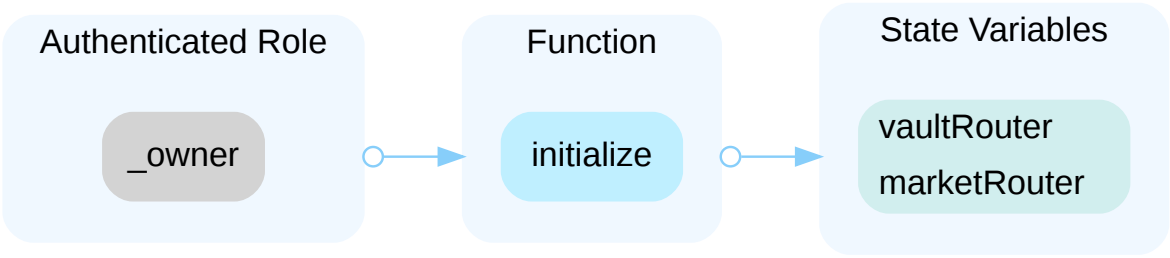
Any compromise to the `MARKET_MGR_ROLE` account may allow the hacker to take advantage of this authority and upgrade market logic implementation contracts to drain user funds.

In the contract `MarketFactory` the role `MARKET_MGR_ROLE` has authority over the functions below.

- function remove(): remove a market from `MarketFactory`.
- function grantPosKeeper(): grant `ROLE_POS_KEEPER` of markets to accounts.
- function marketAskForControllerRole(): grant `ROLE_CONTROLLER` of `grantees` to a market.
- function create(): add a market to `MarketFactory` and initialize it.

Any compromise to the `MARKET_MGR_ROLE` account may allow the hacker to take advantage of this authority and manage markets.

In the contract `MarketReader` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and initialize `MarketReader` contract.

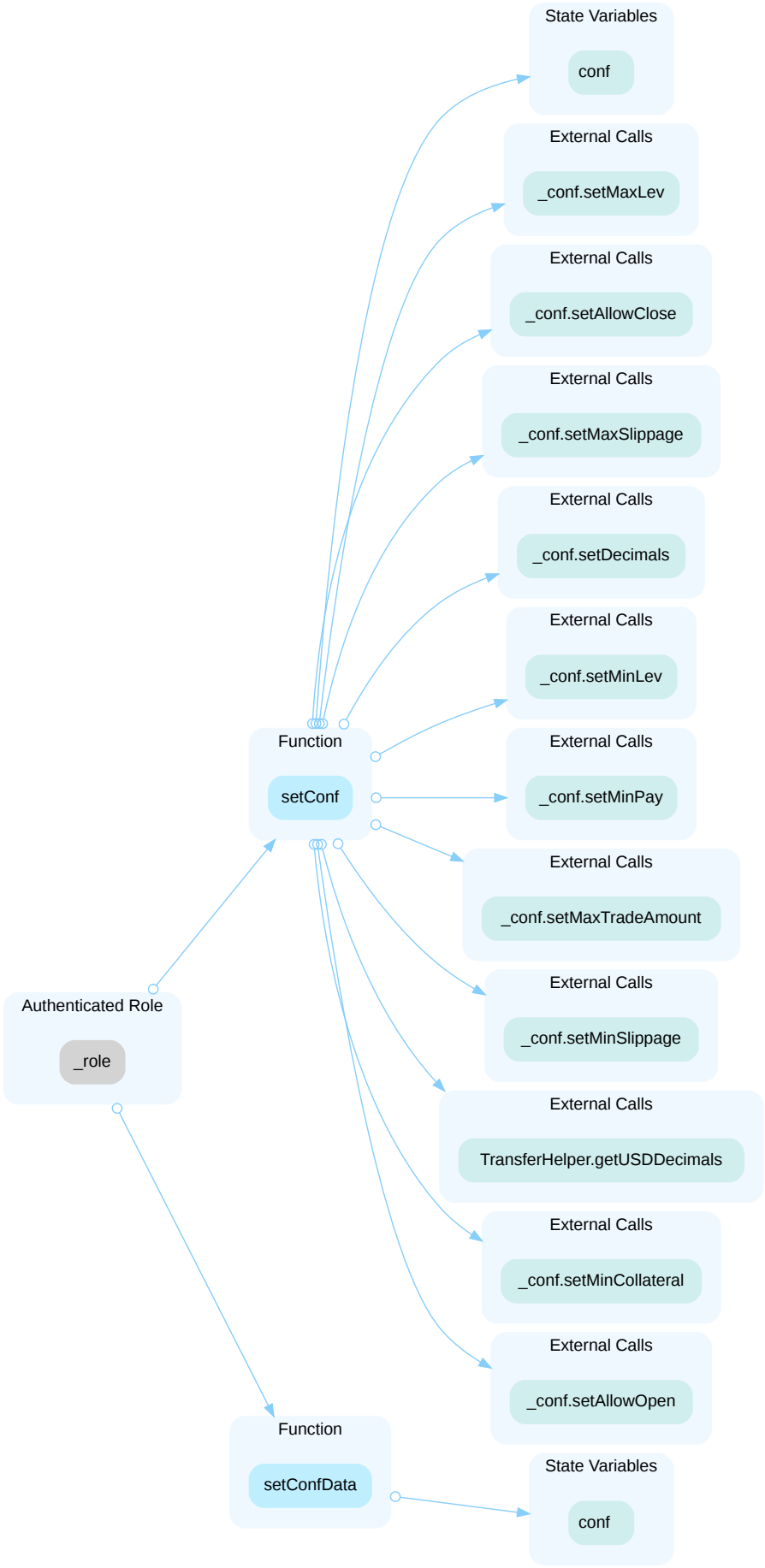


In the contract `MarketRouter` the role `MARKET_MGR_ROLE` has authority over the functions below.

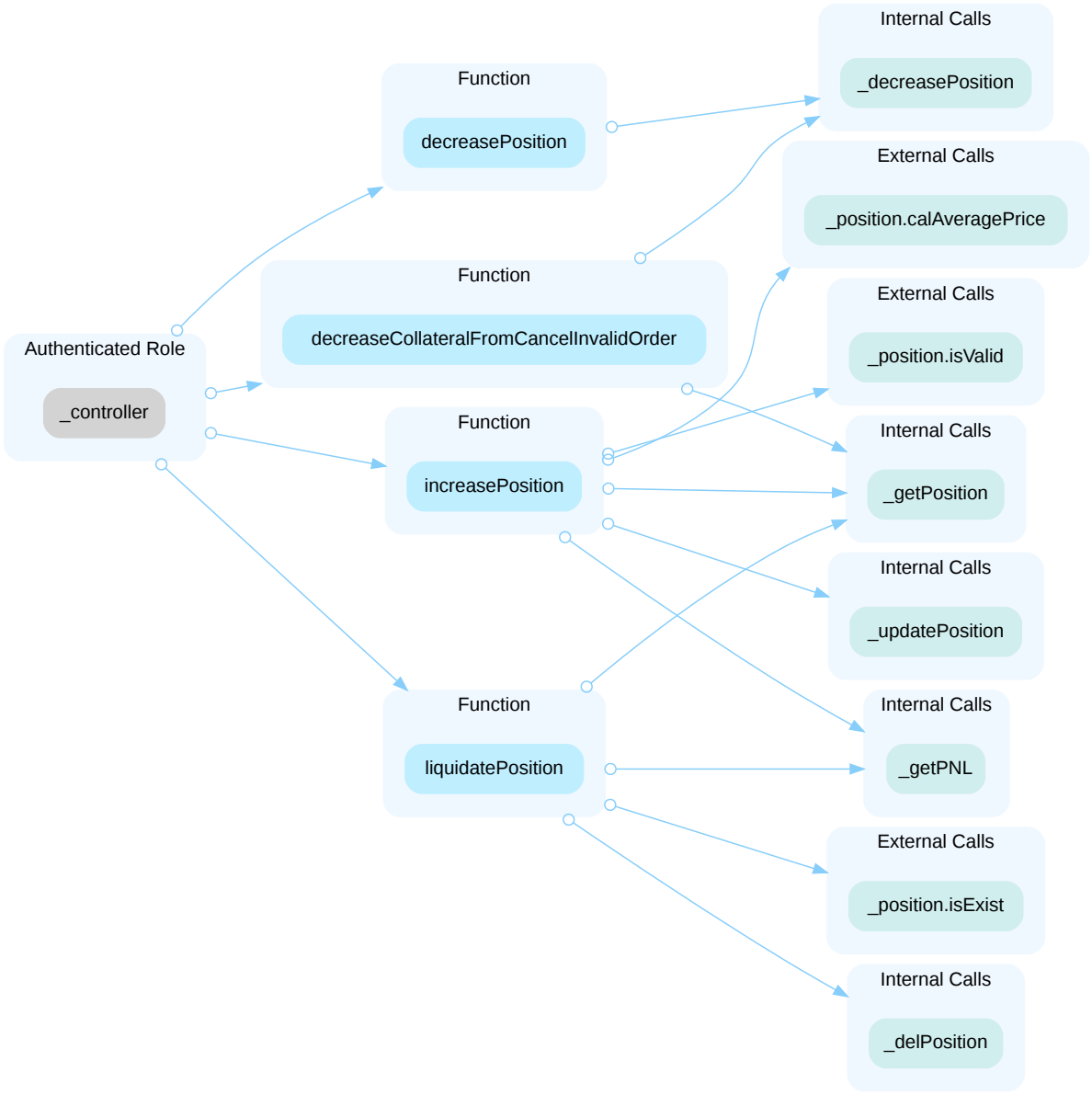
- function addMarket(): add a market.
- function removeMarket(): remove a market.
- function updatePositionBook(): called by `Market.setPositionBook()` to upgrade position book implementation.

Any compromise to the `MARKET_MGR_ROLE` account may allow the hacker to take advantage of this authority and manage markets.

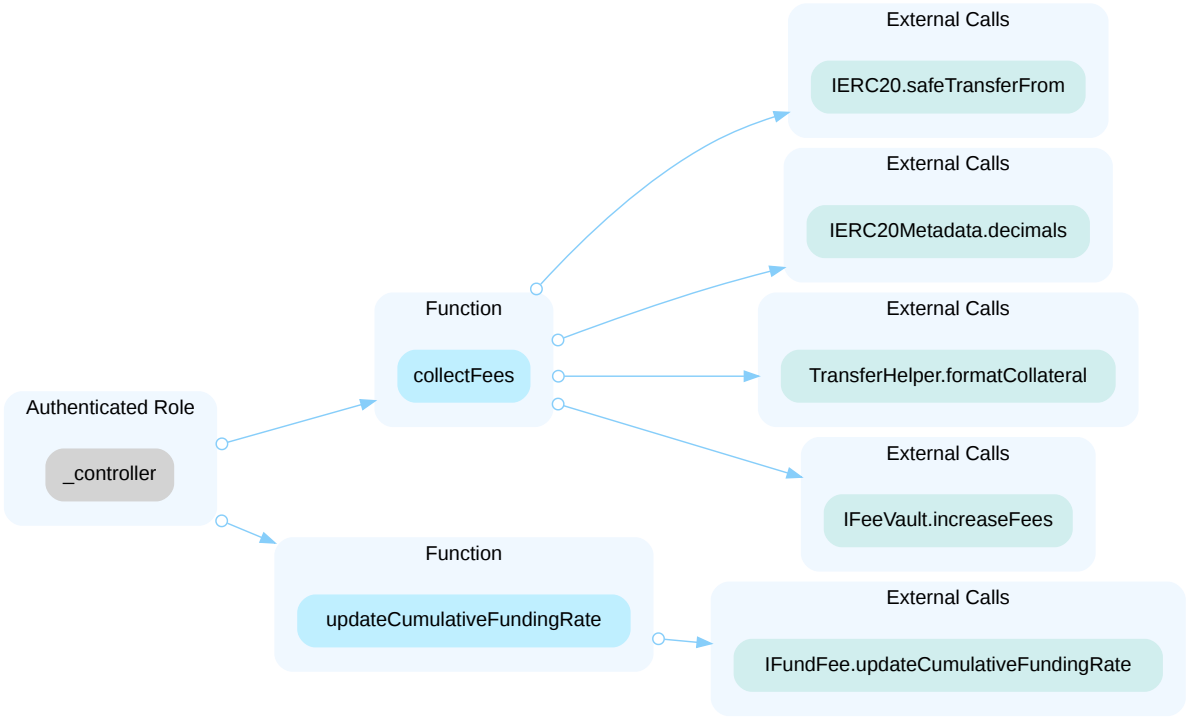
In the contract `MarketValid` the role `MARKET_MGR_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MARKET_MGR_ROLE` account may allow the hacker to take advantage of this authority and set configurations for a market.


State Variables
conf
External Calls
_conf.setMaxLev
External Calls
_conf.setAllowClose
External Calls
_conf.setMaxSlippage
External Calls
_conf.setDecimals
External Calls
_conf.setMinLev
Function
setConf
External Calls
_conf.setMinPay
External Calls
_conf.setMaxTradeAmount
External Calls
_conf.setMinSlippage
Authenticated Role
_role
External Calls
TransferHelper.getUSDDecimals
External Calls
_conf.setMinCollateral
External Calls
_conf.setAllowOpen
Function
setConfData
State Variables
conf

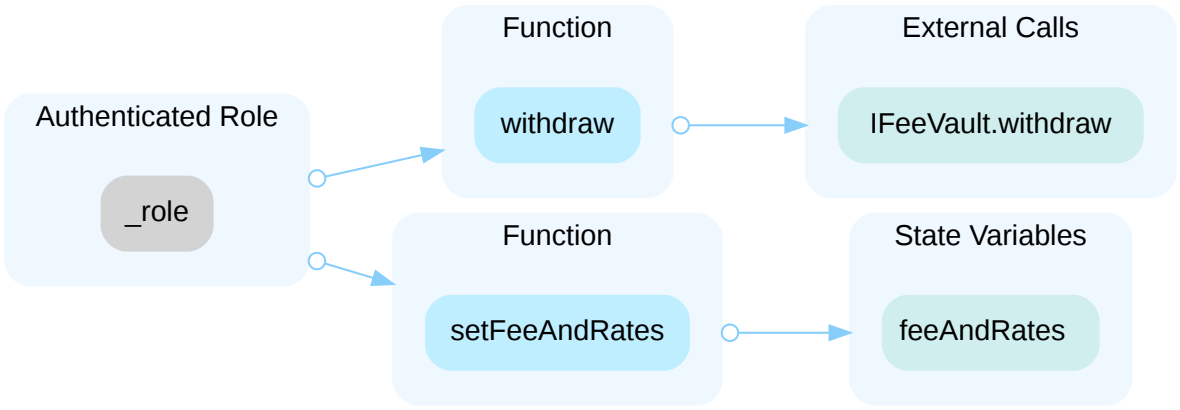
In the contract `PositionBook` the role `ROLE_CONTROLLER` has authority over the functions shown in the diagram below. Any compromise to the `ROLE_CONTROLLER` account may allow the hacker to take advantage of this authority to decrease, increase and liquidate positions.



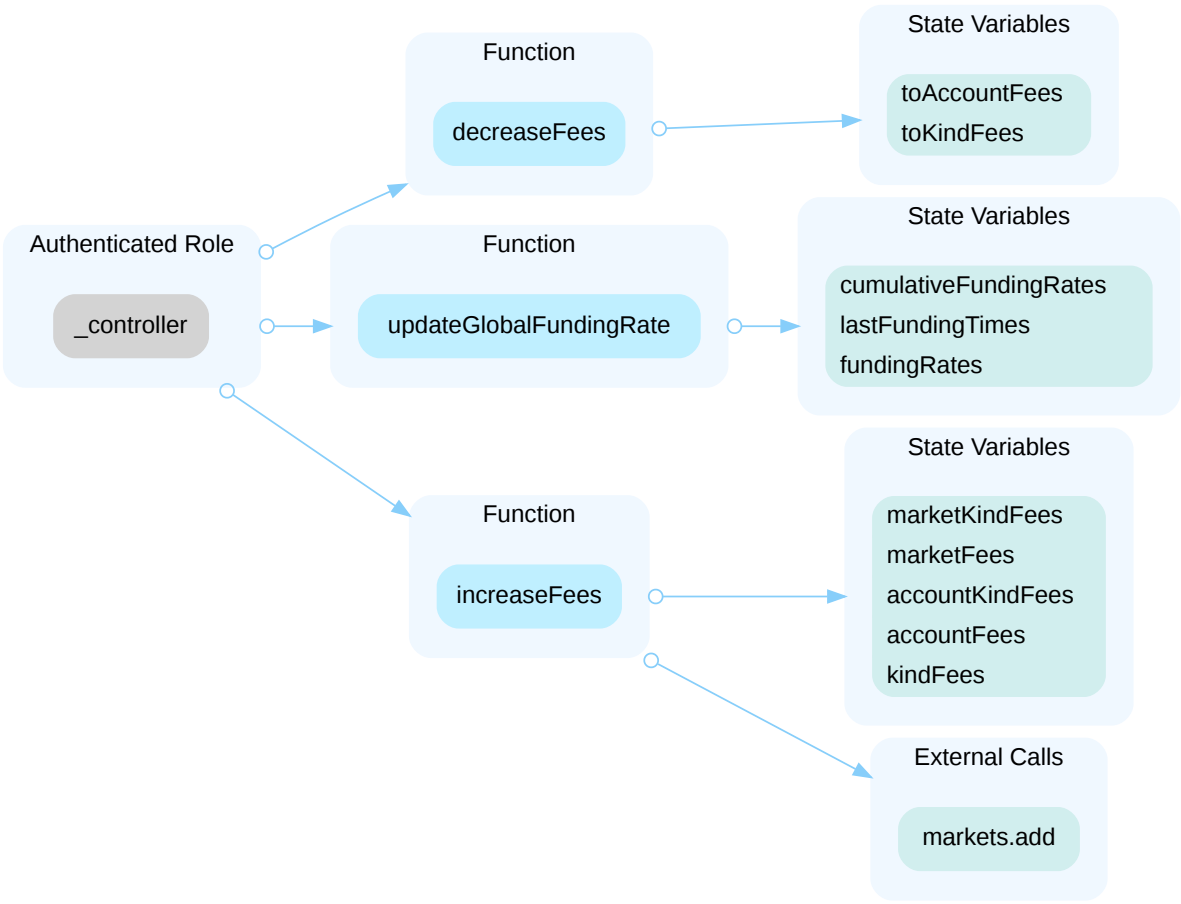
In the contract `FeeRouter` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and change fees/rates at will.



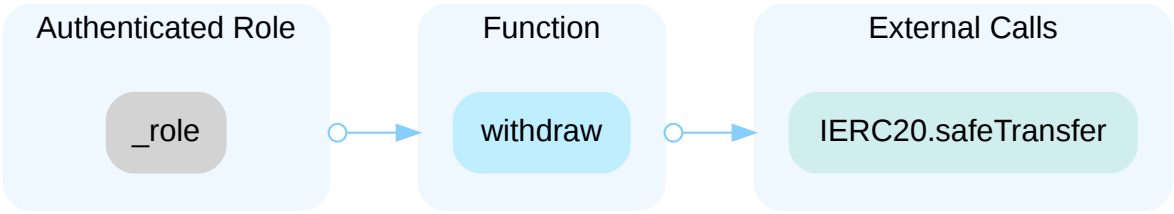
In the contract `FeeRouter` the role `WITHDRAW_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `WITHDRAW_ROLE` account may allow the hacker to take advantage of this authority and steal tokens from the fee router contract.



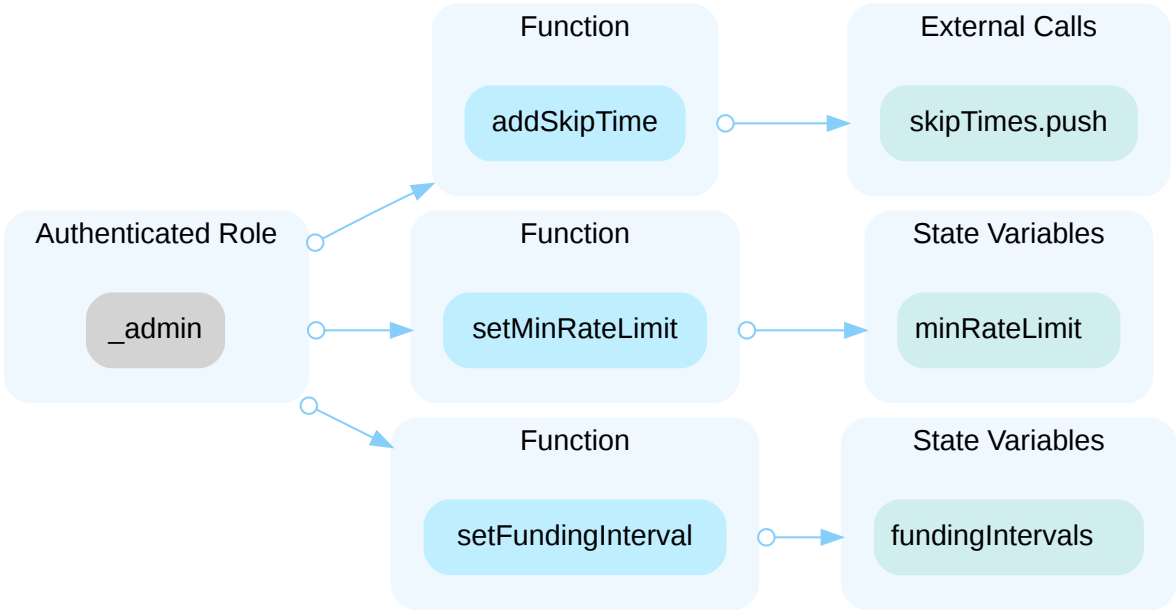
In the contract `FeeVault` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and change fees and funding rate at will.



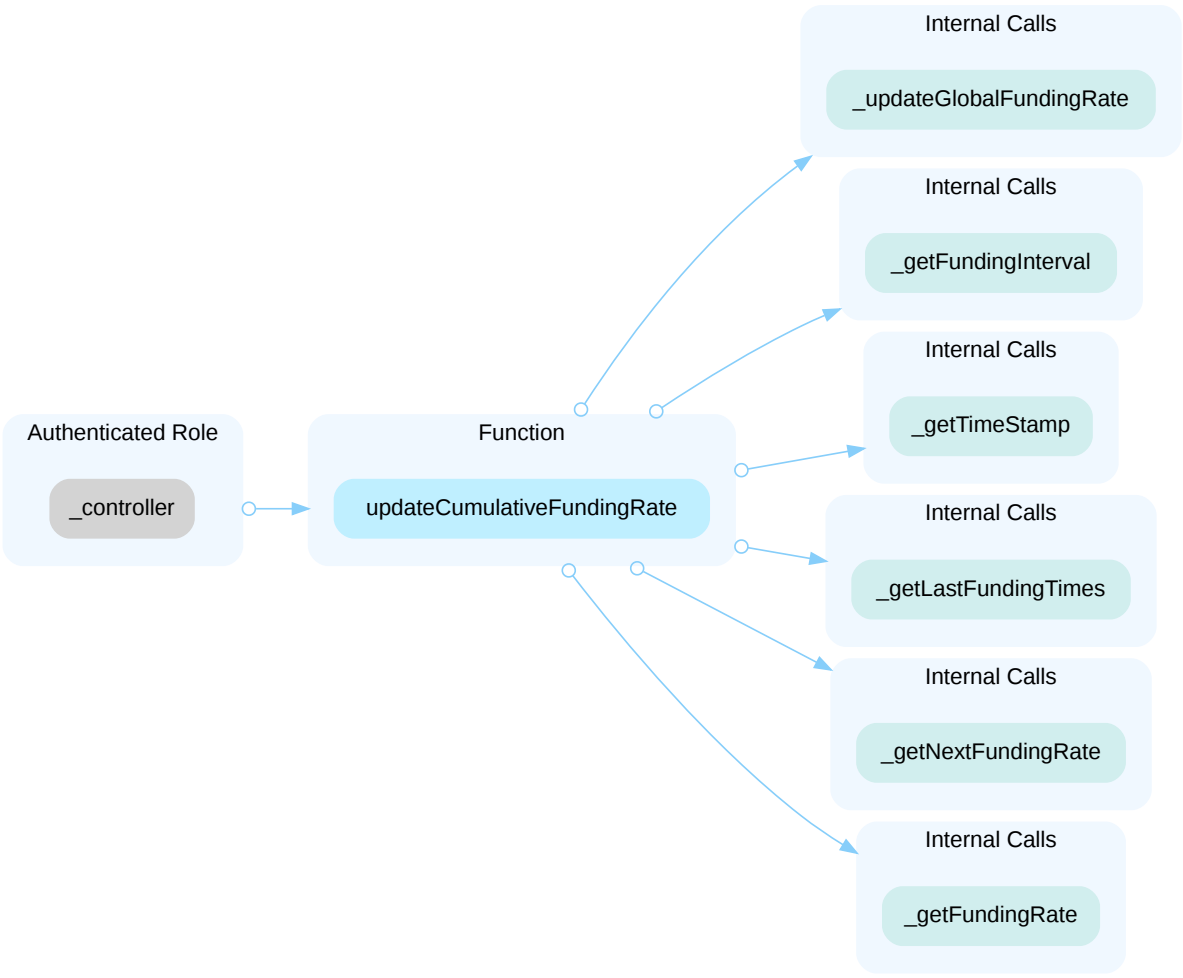
In the contract `FeeVault` the role `WITHDRAW_ROLE` has authority over the functions shown in the diagram below. Any compromise to the WITHDRAW_ROLE` account may allow the hacker to take advantage of this authority and steal tokens from the fee vault.



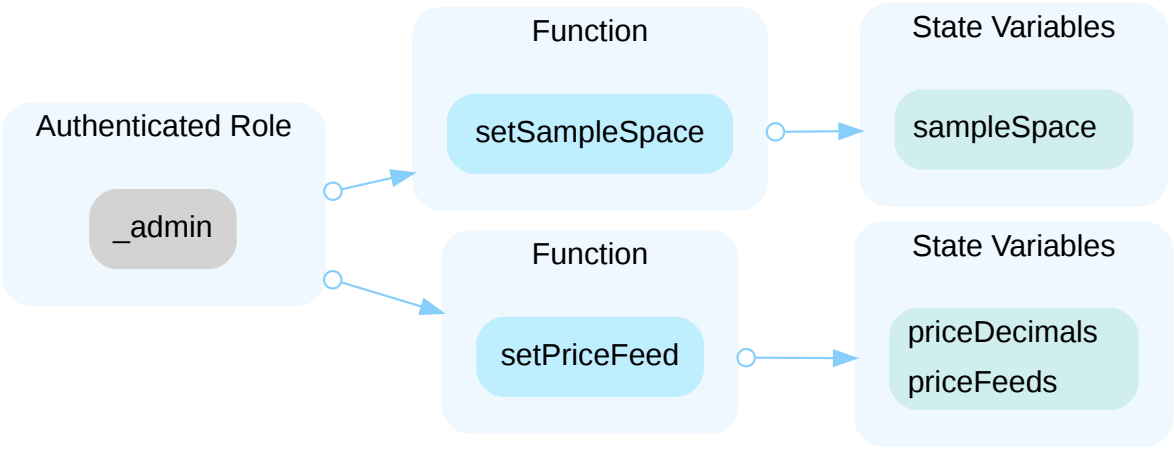
In the contract `FundFee` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and change rate limit, funding interval and skip times at will.



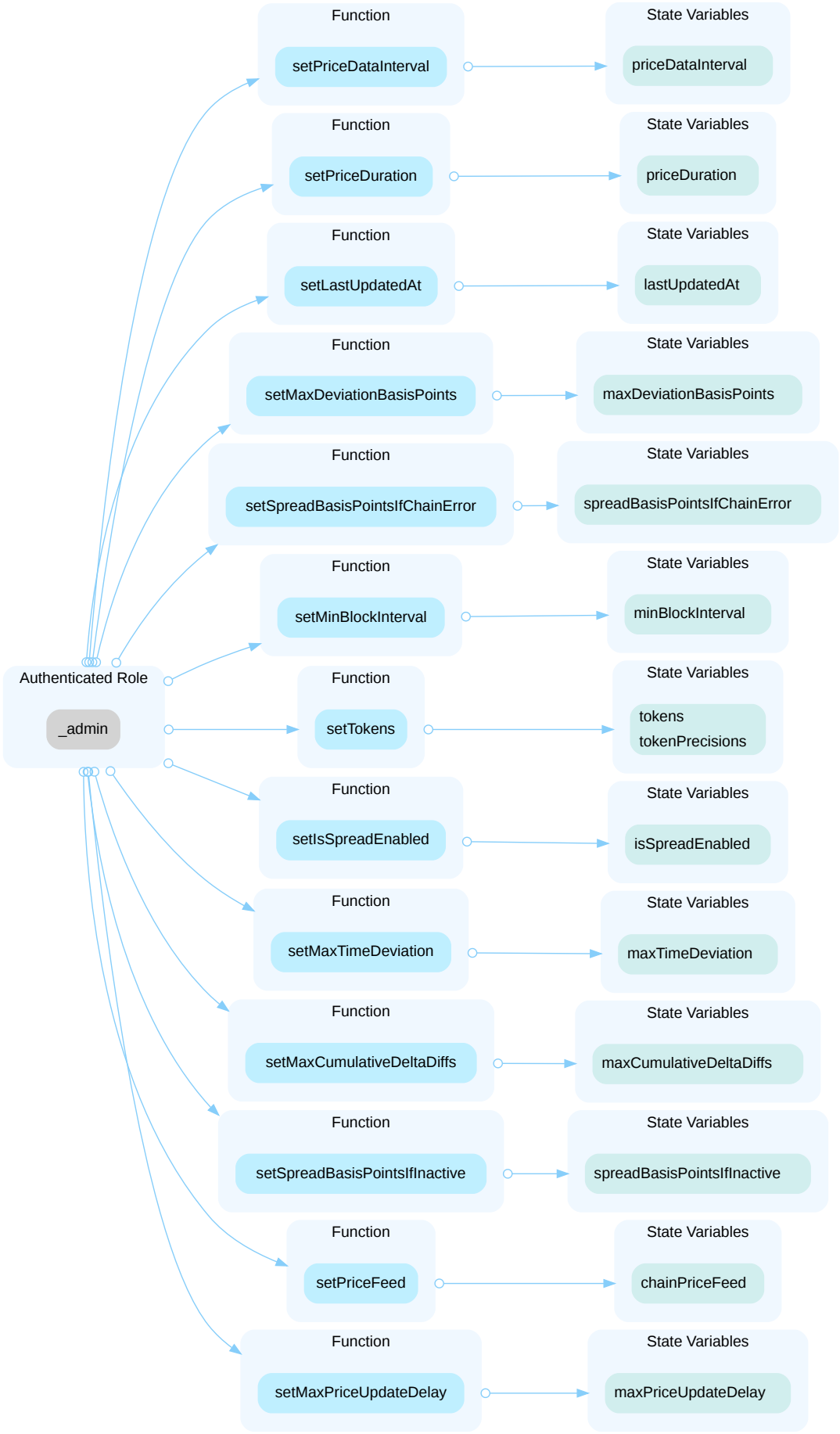
In the contract `FundFee` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and change funding rate at will.



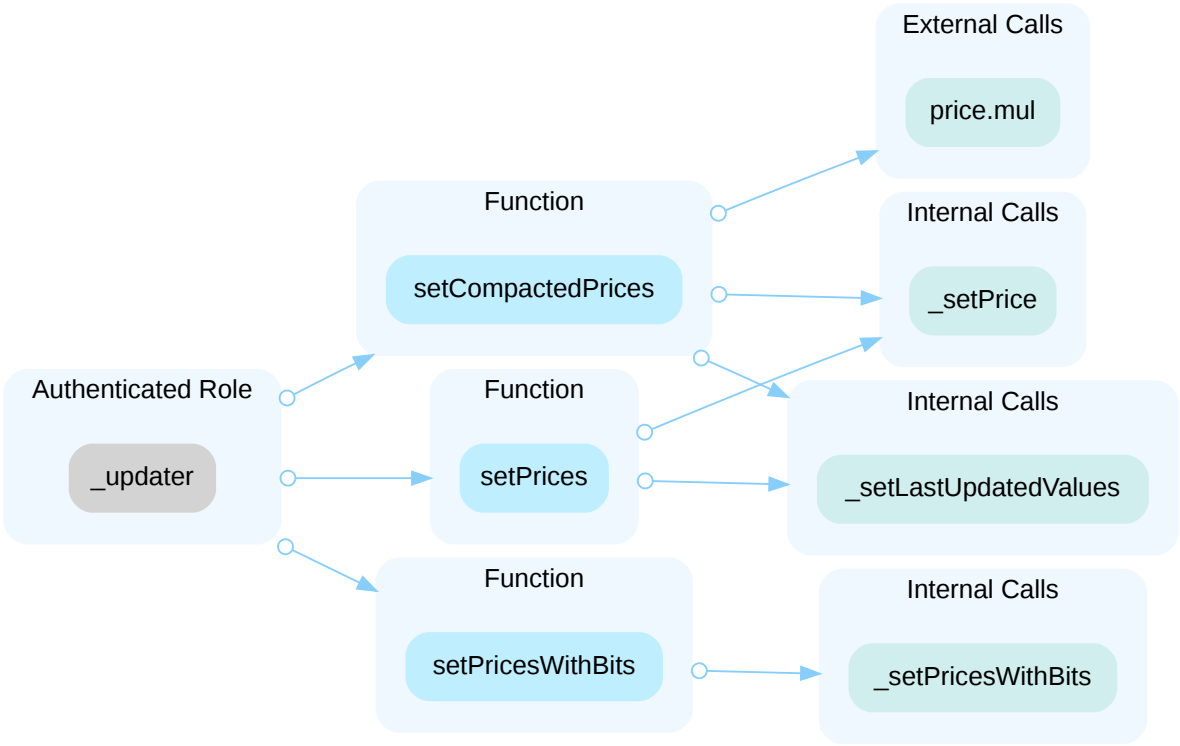
In the contract `ChainPriceFeed` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and manipulate token prices at will.



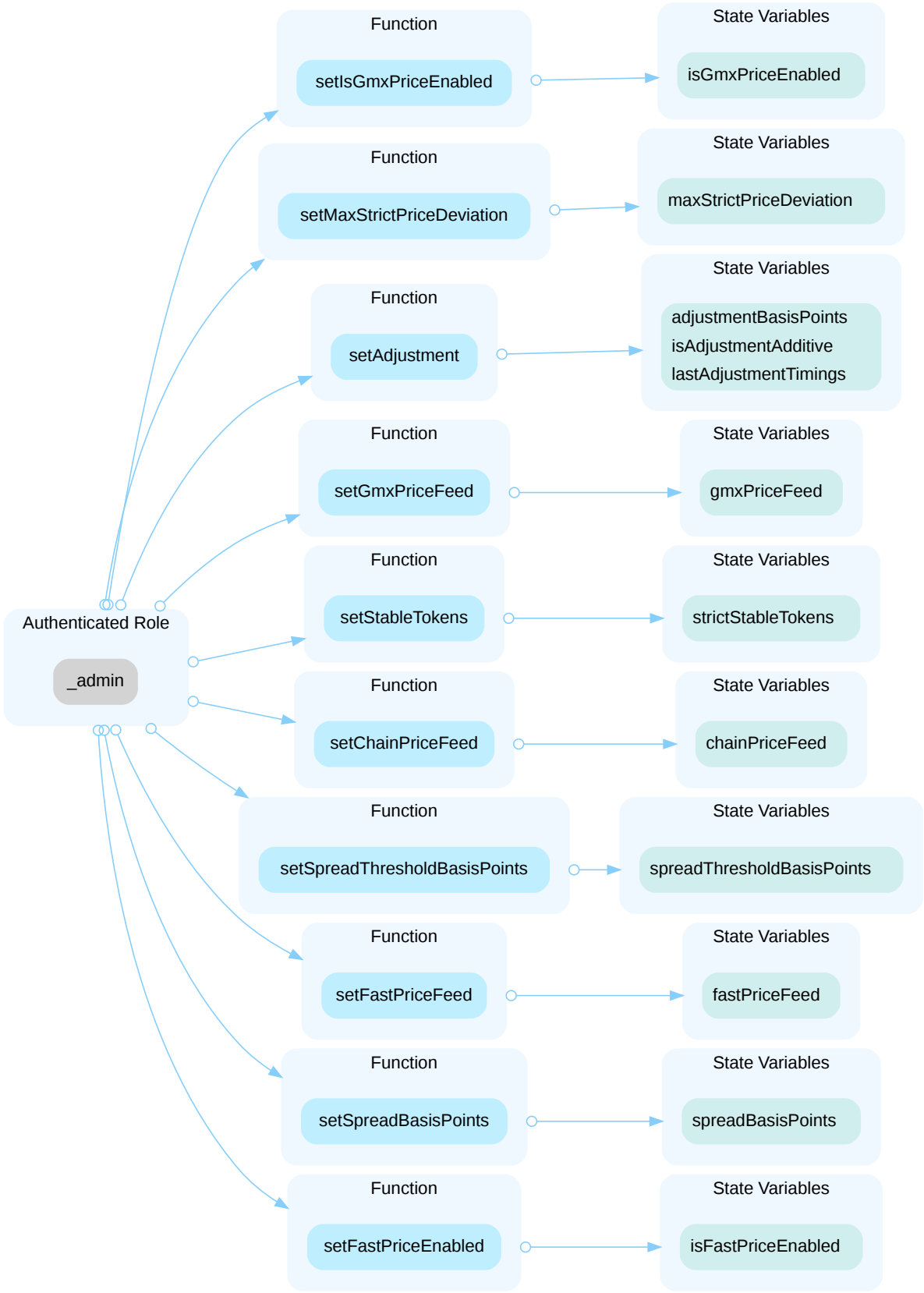
In the contract `FastPriceFeed` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and manipulate token prices at will.


Function
setPriceDataInterval
State Variables
priceDataInterval
Function
setPriceDuration
State Variables
priceDuration
Function
setLastUpdatedAt
State Variables
lastUpdatedAt
Function
setMaxDeviationBasisPoints
State Variables
maxDeviationBasisPoints
Function
setSpreadBasisPointsIfChainError
State Variables
spreadBasisPointsIfChainError
Function
setMinBlockInterval
State Variables
minBlockInterval
Authenticated Role
_admin
Function
setTokens
State Variables
tokens
tokenPrecisions
Function
setIsSpreadEnabled
State Variables
isSpreadEnabled
Function
setMaxTimeDeviation
State Variables
maxTimeDeviation
Function
setMaxCumulativeDeltaDiffs
State Variables
maxCumulativeDeltaDiffs
Function
setSpreadBasisPointsIfInactive
State Variables
spreadBasisPointsIfInactive
Function
setPriceFeed
State Variables
chainPriceFeed
Function
setMaxPriceUpdateDelay
State Variables
maxPriceUpdateDelay

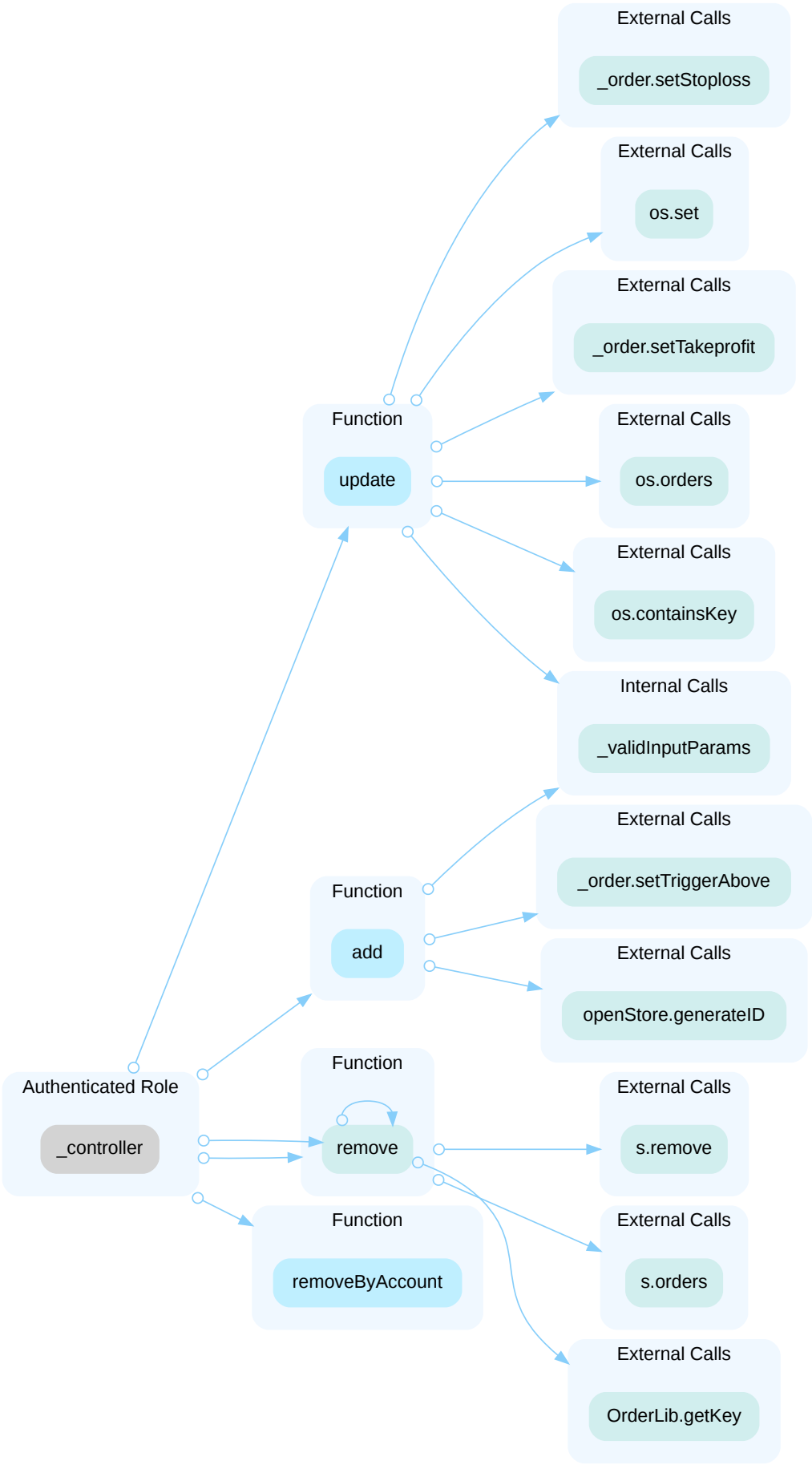
In the contract `FastPriceFeed` the role `_updater` has authority over the functions shown in the diagram below. Any compromise to the `_updater` account may allow the hacker to take advantage of this authority and manipulate token prices at will.



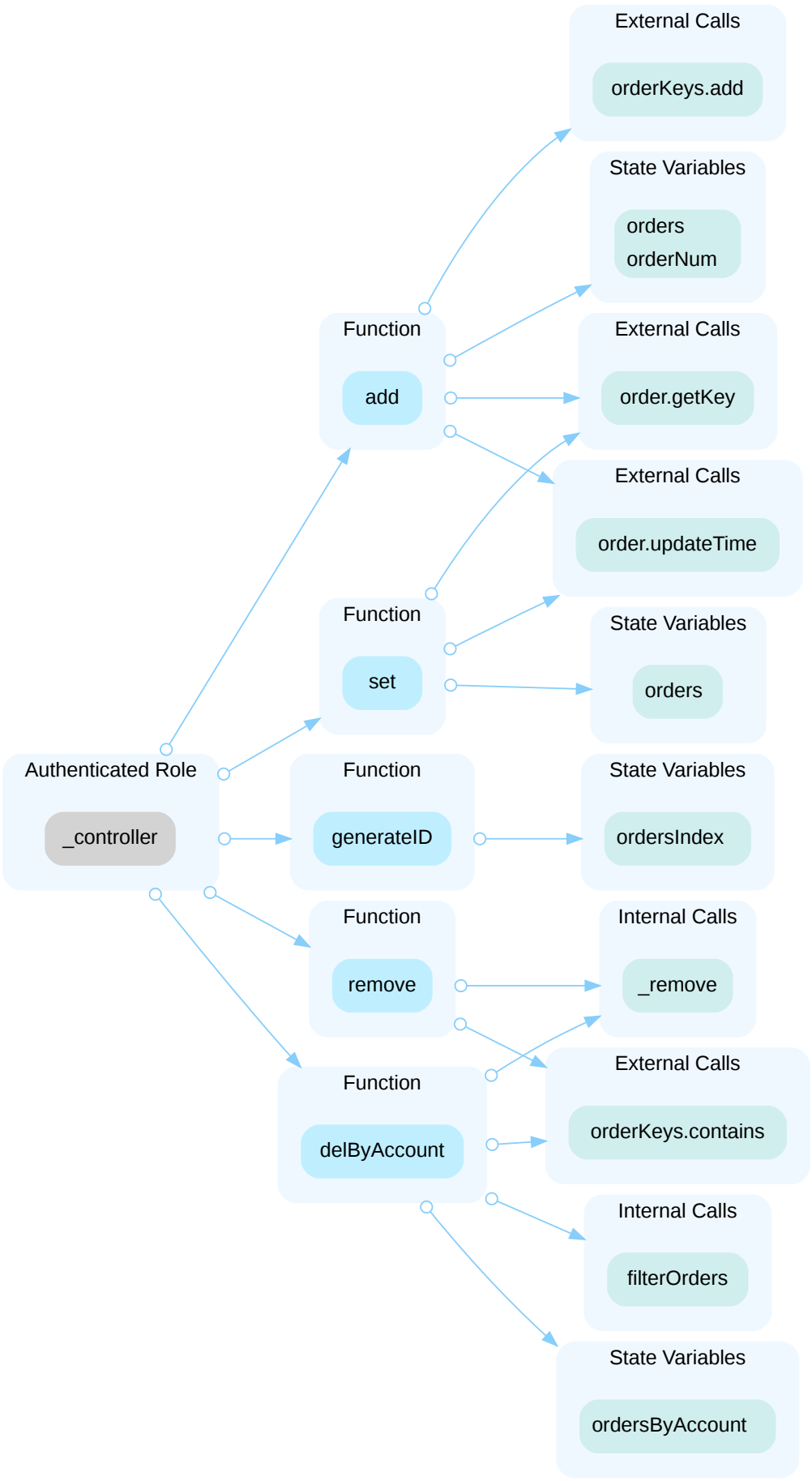
In the contract `Price` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and manipulate token prices at will.



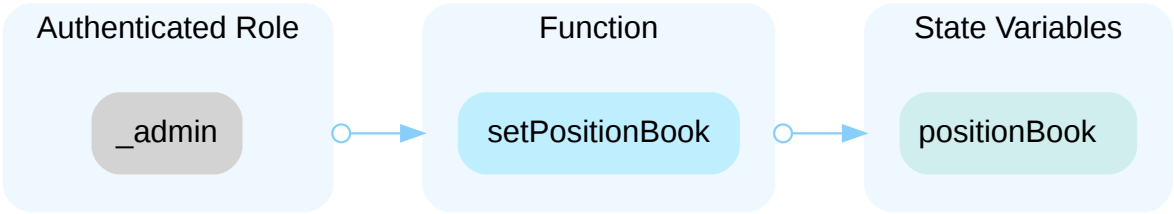
In the contract `OrderBook` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and arbitrarily add/remove/update orders.


External Calls
_order.setStoploss
External Calls
os.set
External Calls
_order.setTakeprofit
Function
update
External Calls
os.orders
External Calls
os.containsKey
Internal Calls
_validInputParams
External Calls
_order.setTriggerAbove
Function
add
External Calls
openStore.generateID
Function
Authenticated Role
_controller
remove
External Calls
s.remove
Function
removeByAccount
External Calls
s.orders
External Calls
OrderLib.getKey

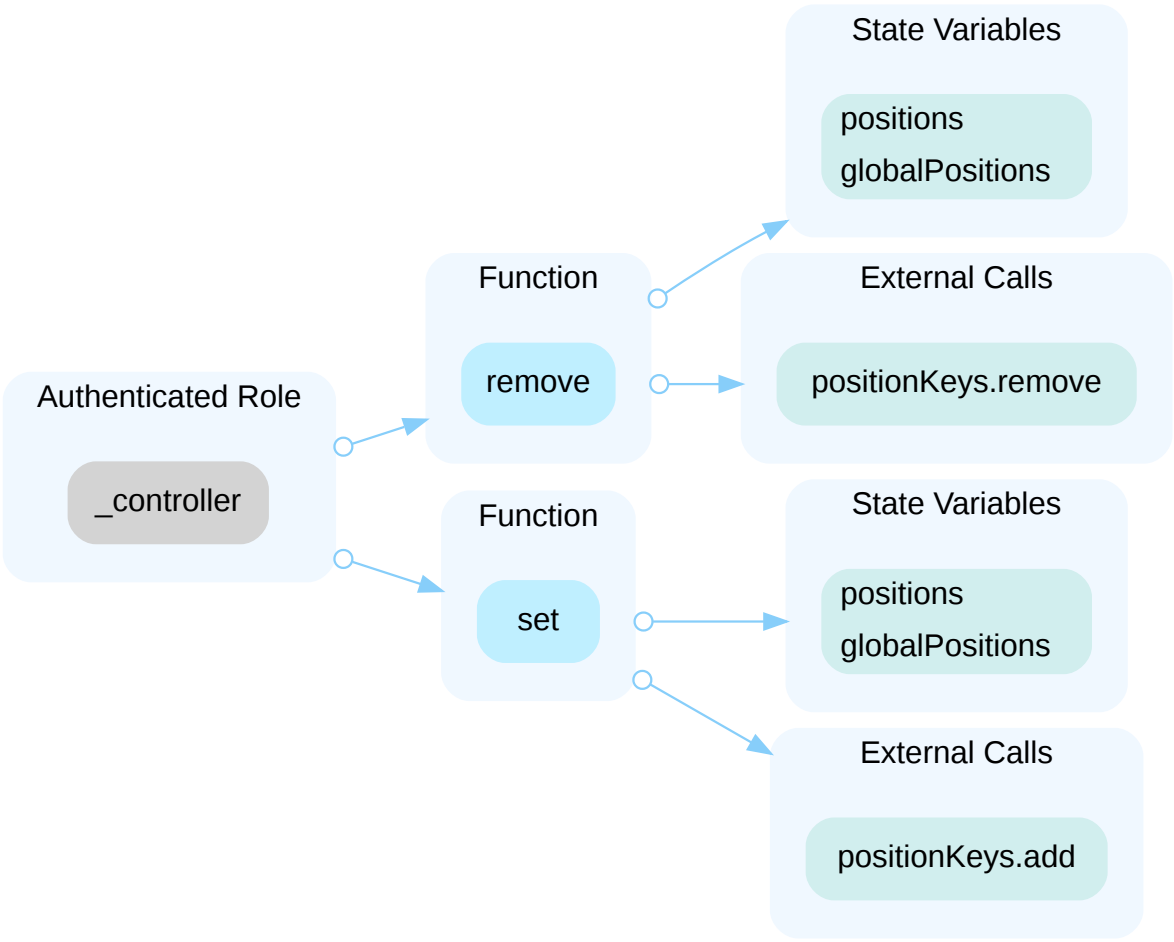
In the contract `OrderStore` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and add/remove/set orders.


External Calls
orderKeys.add
State Variables
orders
orderNum
Function
add
External Calls
order.getKey
External Calls
order.updateTime
Function
set
State Variables
orders
Authenticated Role
_controller
Function
generateID
State Variables
ordersIndex
Function
remove
Internal Calls
_remove
Function
delByAccount
External Calls
orderKeys.contains
Internal Calls
filterOrders
State Variables
ordersByAccount

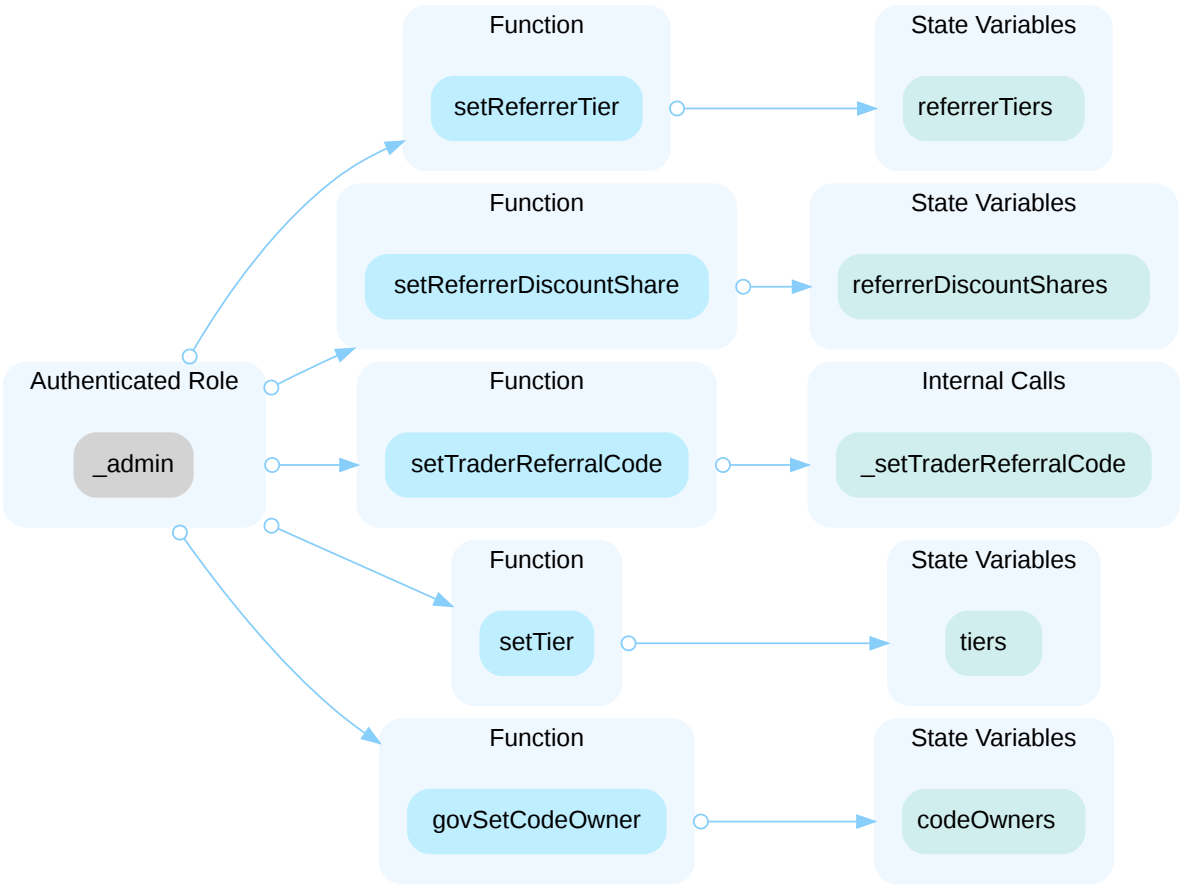
In the contract `PositionStore` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and change position book at will.



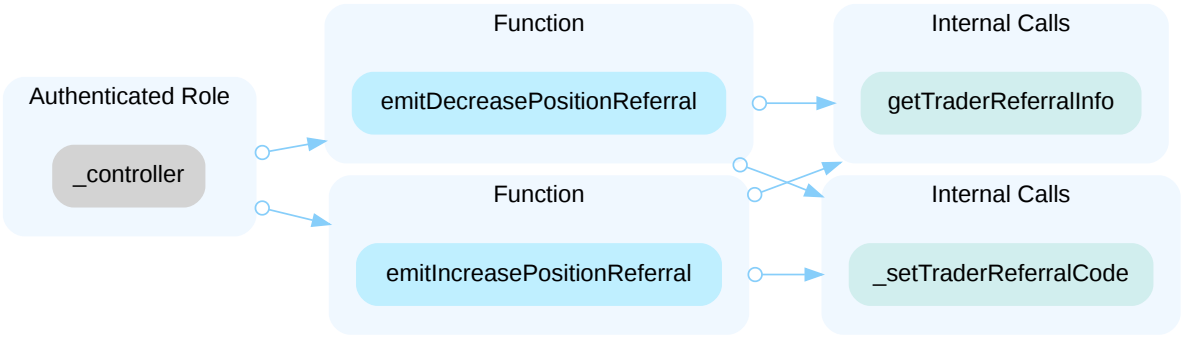
In the contract `PositionStore` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and set/remove accounts at will.



In the contract `Referral` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority and change referral and code information at will.



In the contract `Referral` the role `_controller` has authority over the functions shown in the diagram below. Any compromise to the `_controller` account may allow the hacker to take advantage of this authority and emit referral events arbitrarily.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

No alleviation.

MAR-05 | CONTRACT UPGRADE CENTRALIZATION RISK

Category	Severity	Location	Status
Centralization	**Major**	**contracts/market/Market.sol (v1): 156, 183, 205, 213, 222**	**Acknowledged**

Description

In the contract `Market`, the role `MARKET_MGR_ROLE` has the authority to update market logic implementation contracts behind the `Market` contract.

Any compromise to the `MARKET_MGR_ROLE` account may allow a hacker to take advantage of this authority and control the implementation contracts that are pointed by the market contract, and therefore execute potential malicious functionality in the implementation contract to drain user funds.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key being compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness of privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can fully resolve the risk.

- Renounce the ownership and never claim back the privileged role; OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

The client acknowledged this finding.

CON-06 | SHOULD NOT COLLECT NEGATIVE FEES

Category	Severity	Location	Status
Incorrect Calculation	Medium	contracts/fee/FeeRouter.sol (v1): 111; contracts/market/PositionAddMgr.sol (v1): 192; contracts/market/PositionSubMgr.sol (v1): 213	Resolved

Description

The `_transationsFees()` function is responsible for withdrawing fees from the fee vault. However, the fee router continues to collect fees regardless of whether the total fees are positive or not. The `collectFees()` function performs an incorrect type conversion, resulting in an unexpectedly large number. Consequently, the transaction will be reverted.

```
110         uint256 _amount = TransferHelper.formatCollateral(
111             uint256(_fees), // cast int to uint, not safe.
112             IERC20Metadata(token).decimals()
113         );
```

Proof of Concept

```
contract FeeRouterTest is Test, Ac {
    FeeRouter fr;
    IERC20 public constant TOKEN =
IERC20(0x95aD61b0a150d79219dCF64E1E6Cc01f0B64C4cE);

    constructor() Ac(address(0)) {}

    function setUp() public {
        vm.createSelectFork("mainnet");
        fr = new FeeRouter(address(this));
        fr.grantAndRevoke(ROLE_CONTROLLER, address(this));
        deal(address(TOKEN), address(this), 1_000_000_000 ether);
        TOKEN.approve(address(fr), type(uint256).max);
    }

    function testCollectFees() public {
        int256[] memory fees = new int256[](1);
        fees[0] = -1000 ether;
        console.log(uint(fees[0]));
        // collect fees from address(this) and revert
        fr.collectFees(address(this), address(TOKEN), fees);
    }
}
```

```
Running 1 test for test/FeeRouter.t.sol:FeeRouterTest
[FAIL. Reason: Arithmetic over/underflow] testCollectFees() (gas: 18375)
Logs:
  115792089237316195423570985008687907853269984665640564038457584007913129639936

Traces:
  [18375] FeeRouterTest::testCollectFees()
    ├─ [0] console::f5b1bba9(ffffffffffffffffffffffffffffffffffffffffffffffffffffffffc9ca36523a21600000) [staticcall]
    │   └─ ← ()
    ├─ [9838] FeeRouter::collectFees(FeeRouterTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], TokenMintERC
19dCF64E1E6Cc01f0B64C4cE], [-10000000000000000000])
    │   ├─ [2438] TokenMintERC20Token::decimals() [staticcall]
    │   │   └─ ← 18
    │   └─ ← "Arithmetic over/underflow"
    └─ ← "Arithmetic over/underflow"

Test result: FAILED. 0 passed; 1 failed; finished in 2.92s
```

Recommendation

We recommend not collecting negative fees and using SafeCast to cast all int numbers.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

CVB-05 OLD VAULT ROUTER IS NOT REMOVED FROM `ROLE_CONTROLLER` WHEN SETTING NEW VAULT ROUTER

Category	Severity	Location	Status
Coding Issue	Medium	contracts/vault/CoreVault.sol (v1): 44~45	Resolved

Description

The public function `setVaultRouter()` should remove old vault router from `ROLE_CONTROLLER`. Otherwise, after setting new vault router, old vault router still has `ROLE_CONTROLLER` privilege, which may be misused to call `transferOutAssets()` to steal tokens from the contract.

Recommendation

Recommend removing old vault router from `ROLE_CONTROLLER`.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

FEE-01 | FUNDING RATE IS ALWAYS POSITIVE

Category	Severity	Location	Status
Design Issue	Medium	contracts/fee/FeeRouter.sol (v1): 187~192; contracts/fee/FundFee.sol (v1): 145, 229	Acknowledged

Description

The purpose of a funding rate is to keep the price of the protocol as close to the price of the underlying asset as possible. If one starts trading at a price lower than the underlying asset's, the funding rate should be negative, and those who shorted the asset should pay those who longed it.

However, in the case of a short position, the funding rate fee is calculated by `(cumulativeShortFundingRate - cumulativeEntryShortFundingRate) * size`. As the `cumulativeShortFundingRate` is equal to or greater than `cumulativeEntryShortFundingRate`, the funding rate can never be a negative value. As a result, the funding rate fee will be collected by the fee vault.

PositionSubMgr.sol

```
153         int256[] memory _fees = feeRouter.getFees(_params, _position);
```

FeeRouter.sol

```
187         int256 _fundFee = _getFundingFee(
188             _market,
189             params._isLong,
190             position.size,
191             position.entryFundingRate
192         );
```

FundFee.sol

```
155         int256 _cumRates = IFeeVault(feeVault).cumulativeFundingRates(
156             market,
157             isLong
158         );
```

Recommendation

We recommend revisiting the funding rate design.

Alleviation

[Blex Team]: The rules of this product design may indeed be negative, but they are all positive for the trader, because they all need to pay the capital fee to the LP.

GVB-01 USEABLE NET SIZE CALCULATION WITHOUT `isLong` CHECK

Category	Severity	Location	Status
Logical Issue	Medium	contracts/market/GlobalValid.sol (v1): 127, 140	Resolved

Description

The `_getMaxUseableNetSize()` function calculates the maximum usable position size based on the delta net size and the `maxNetSizeLimit` using the formula `maxNetSizeLimit - deltaNetSize`. If `shortSize > longSize` and the delta net size reaches the limit, it indicates that the short position number has reached its limit and new short positions should not be opened. However, in this case, increasing long positions is also not allowed. If `shortSize < longSize`, the opposite applies.

In addition, the `_getMaxUseableUserNetSize()` function that gets the user's net size has the same issue.

Recommendation

We recommend handling the aforementioned scenario.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MAK-01 | INPUT `isOpen` IS NOT SET FOR `decreasePosition()`

Category	Severity	Location	Status
Volatile Code	Medium	contracts/market/MarketValid.sol (v1): 56~64; contracts/market/PositionSubMgr.sol (v1): 42	Resolved

Description

The input `_vars.isOpen` for `decreasePosition()` function is not explicitly set to "false". If it is set to "true", the `mv.validPosition()` function will use the incorrect input "1"/"2" for `busType` parameter of `validCollateralDelta()` function.

```
166             mv.validPosition(_params, _position, _fees);
```

MarketValid.sol

```
56          if (_params.isOpen) {
57              validCollateralDelta(
58                  _params.collateralDelta > 0 ? 1 : 2,
59                  _position.collateral,
60                  _params.collateralDelta,
61                  _position.size,
62                  _params._sizeDelta,
63                  _fees.totoalFees()
64              );
```

As a result, the leverage validation could be incorrect.

Recommendation

We recommend setting the input `_vars.isOpen` to true for `decreasePosition()` function.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MVB-04 | NOT BE ABLE TO `getCollateralRange()`

Category	Severity	Location	Status
Incorrect Calculation	Medium	contracts/market/MarketValid.sol (v1): 223	Resolved

Description

In the `getCollateralRange()` function, if `_isIncrease` is true, the valid `_oldCollertal` could be greater than collateral number at the maximum leverage. In this case, `(_sizeDelta + _oldSize) / conf.getMaxLev()` is smaller than `_oldCollertal`, and subtracting from it will result in an arithmetic error.

Proof of Concept

```
contract MarketValidTest is Test, Ac {
    MarketValid mv;

    constructor() Ac(address(0)) {}

    function setUp() public {
        mv = new MarketValid(address(this));
        mv.grantAndRevoke(MARKET_MGR_ROLE, address(this));
        // leverage range: 1 ~ 10
        mv.setConf(1, 100, 1, 10, 100_000, 100, 1, true, true, 0);
    }

    function testGetCollateralRange() public view {
        (uint256 maxCollateralDelta, uint256 minCollateralDelta) =
mv.getCollateralRange(true, 50 ether, 100 ether, 0);
        console.log(maxCollateralDelta);
        console.log(minCollateralDelta);
    }
}
```

```
Running 1 test for test/MarketValid.t.sol:MarketValidTest
[FAIL. Reason: Arithmetic over/underflow] testGetCollateralRange() (gas: 8331)
Traces:
  [8331] MarketValidTest::testGetCollateralRange()
    ├─ [3226] MarketValid::getCollateralRange(true, 5000000000000000000, 10000000000000000000, 0) [staticcall]
    │   └─ ← "Arithmetic over/underflow"
    └─ ← "Arithmetic over/underflow"

Test result: FAILED. 0 passed; 1 failed; finished in 15.31ms
```

Recommendation

We recommend fixing this issue.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MVB-05 | INCONSISTENT LEVERAGE CALCULATION

Category	Severity	Location	Status
Inconsistency	Medium	contracts/market/MarketValid.sol (v1): 107~111, 336	Resolved

Description

In the `validCollateralDelta()` function, the leverage is calculated by including fees: `lev = newSize / newCollateral = newSize / (_collateral +/- _collateralDelta +/- _fees)`

However, in the `validateLiquidation()` function, the `remainingCollateral` used to check leverage does not include fees.

Recommendation

We recommend implementing leverage checks with the same logic.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

OBB-02 | INCONSISTENT ORDER QUERY CONDITION

Category	Severity	Location	Status
Inconsistency	Medium	contracts/order/OrderBook.sol (v1): 54, 72	Resolved

Description

The first order query condition contains `key != bytes32(0)` while the second order query condition does not. So it is possible that the counted orders in the second query are not in the first query and returned orders' keys may be zero.

Recommendation

Recommend reviewing order query conditions and make sure they are consistent.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

OMB-01 CHARGE `execFee` ONLY WHEN THE ORDER HAS SUFFICIENT COLLATERAL

Category	Severity	Location	Status
Logical Issue	Medium	contracts/market/OrderMgr.sol (v1): 191	Resolved

Description

If order execution fails, the `sysCancelOrder()` function will be called to delete the order and charge `execFee` if it is an increase order.

1. The market approves `execFee` to be sent to `feeRouter`.
2. The `feeRouter` collects fees.
3. If `_isTransferToUser` is true, remaining collateral will be transferred to the order account.

However, if the value of `_order.collateral` is smaller than the `execFee`, the `feeRouter` ends up taking more tokens than the order initially escrowed.

Recommendation

We recommend checking `_order.collateral >= execFee` before charging the `execFee`.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

OMB-02 DECREASE COLLATERAL WITHOUT CHECKING LIQUIDATION

Category	Severity	Location	Status
Logical Issue	Medium	contracts/market/OrderMgr.sol (v1): 216	Acknowledged

Description

In the `_cancelOrder()` function, when dealing with a decrease order and the `isExec` flag is set to true, because the order itself does not escrow any collateral, it reduces the collateral to charge the `execFee` . However, there is no check to ensure that the position size does not exceed the maximum leverage limit.

Recommendation

Consider adding the check.

Alleviation

[Blex Team]: Issue acknowledged. I won't make any changes for the current version.

PAM-01 CHECK `collateralDelta` INPUT BEFORE UPDATING POSITIONS

Category	Severity	Location	Status
Volatile Code	Medium	contracts/market/PositionAddMgr.sol (v1): 241	Resolved

Description

The `execOrderKey()` function can be called by the "ROLE_POS_KEEPER" role to update positions. It does not deposit any collateral tokens, so the input `_params.collateralDelta` should be equal to `order.collateral`. If their values do not match, it will add the specified collateral amount to the position without any payment.

Recommendation

We recommend adding the aforementioned check.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-01 UNABLE TO LIQUIDATE DUE TO INCONSISTENT FEE CHARGES

Category	Severity	Location	Status
Inconsistency	Medium	contracts/market/PositionSubMgr.sol (v1): 213, 272	Resolved

Description

If the total fee amount exceeds the collateral of a position, it indicates that the position is unable to cover the fees. It approves the collateral number for the `feeRouter`, which subsequently charges fees.

```
268                 uint256 amount = TransferHelper.formatCollateral(
269                     uint256(_position.collateral),
270                     IERC20Decimals(collateralToken).decimals()
271                 );
272                 IERC20(collateralToken).approve(address(feeRouter), amount);
// approved collateral amount, but not total fees.
```

However, the `_fees` to be charged by the `feeRouter` are not updated to reflect this scenario. Consequently, the `feeRouter` transfers more tokens than the approved number and the transaction will fail.

```
213         feeRouter.collectFees(_params._account, collateralToken, _fees);
```

Recommendation

We recommend charging the approved fee amount.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-06 | UNFAIR DESIGN FOR VAULT INVESTORS

Category	Severity	Location	Status
Design Issue	Medium	contracts/market/PositionSubMgr.sol (v1): 248	Acknowledged

Description

The vault serves as a liquidity provider for the market.

When decreasing or liquidating a position, the market withdraws the position's profits from the vault and transfers them to the position account. Conversely, if the position incurs losses, the market transfers the collateral of the position back to the vault. However, while the market is always able to withdraw tokens from the vault to cover the position's profits (if the vault balance is sufficient), in cases where the position's collateral is insufficient to cover losses and fees, the market only transfers all the collateral to the vault. As a result, the vault receives a lower amount of tokens than it should in this scenario.

In summary, if a position is liquidated and its collateral cannot cover fees and losses, the vault ends up earning less, which is unfair for vault investors.

Recommendation

Consider adding a funding mechanism to compensate the vault when a position lacks sufficient collateral.

Alleviation

[Blex Team]: Issue acknowledged. I won't make any changes for the current version.

PSM-10 | LACK OF TRIGGER PRICE VALIDATION

Category	Severity	Location	Status
Volatile Code	Medium	contracts/market/PositionSubMgr.sol (v1): 395	Acknowledged

Description

The protocol relies on position keeper roles to run an off-chain service, monitoring orders to be executed. However, the `PositionSubMgr.execOrderKey()` function lacks a check to ensure that the mark price hits the trigger price of orders. If the off-chain service does not work as expected, orders executed by the position keeper may not follow the order price rules.

Recommendation

Consider adding a check in the contract to ensure the mark price is valid before executing an order.

Alleviation

[Blex Team]: The change method is only called when the system reduces the warehouse and will not be used by the user. When the system judges that the warehouse pledge is not enough, the method is called to perform the reduction. The `decreasePositionFromOrder()` function calls the `_getClosePrice()` function to fetch the price from Chainlink and Fastprice in order to compare it for the current chain of positions, so there is no need to price.

VAU-01 | UNPROTECTED `initialize()` FUNCTION

Category	Severity	Location	Status
Coding Issue	Medium	contracts/vault/CoreVault.sol (v1): 38; contracts/vault/RewardDistributor.sol (v1): 26~29; contracts/vault/VaultReward.sol (v1): 35~40; contracts/vault/VaultRouter.sol (v1): 37~40	Acknowledged

Description

The `initialize()` function can be called by anyone. It is possible that after the contract is deployed, a hacker calls `initialize()` with malicious `rewardToken/rewardTracker/vaultRouter/...` parameters before the deployer does it. Then if the deployer does not notice that his/her initialize transaction fails and continues using the hacked contract, the hacker may be able to steal tokens from the contract.

Recommendation

Recommend adding access control to `initialize()` or move the code in `initialize()` to constructor.

Alleviation

From BLEX team: "Thank you for the notice. We will review the failure information when deploying contracts. However, the code will not be modified for this issue."

VRB-01 REWARD CALCULATION MAY BE INCORRECT WHEN USERS CALL VAULT/VAULTROUTER BUY()/SELL()/DEPOSIT()/WITHDRAW() DIRECTLY

Category	Severity	Location	Status
Logical Issue	Medium	contracts/vault/VaultReward.sol (v1): 68, 87	Acknowledged

Description

Users' vault deposit rewards are only maintained and calculated in VaultReward contract. But it is possible for users to call buy()/sell()/deposit()/withdraw() directly in CoreVault/VaultRouter contracts. In this case, users will not get any reward for their vault deposit.

Proof of Concept

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol"; import {ICoreVault, IERC4626} from "./interfaces/ICoreVault.sol"; import "./interfaces/IVaultRouter.sol"; import "./interfaces/IVaultReward.sol";

contract Poc { IERC4626 vault; IVaultReward vaultReward; IERC20 rewardToken; IVaultRouter vaultRouter;

```
constructor(IERC4626 _vault, IVaultReward _vaultReward, IERC20 _rewardToken,
IVaultRouter _vaultRouter) {
    vault = _vault;
    vaultReward = _vaultReward;
    rewardToken = _rewardToken;
    vaultRouter = _vaultRouter;
}

function buy() public {
    IERC20 asset = IERC20(vault.asset());
    asset.approve(vaultRouter, asset.balanceOf(address(this)));
    uint256 shares = vaultRouter.buy(vault, address(this),
asset.balanceOf(address(this)), 0);
}

function claimReward() public {
    vaultReward.claimLPReward();
    require(rewardToken.balanceOf(address(this)) != 0, "no reward received!");
}
```

}

1. transfer some vault asset tokens to Poc contract
2. call Poc buy() function
3. wait for enough time to earn rewards
4. call Poc claimReward() function: the call will revert because no reward is received.

Recommendation

Recommend restricting access to vault/vaultRouter buy()/sell()/deposit()/withdraw() such that they can only be called by VaultReward.

Alleviation

No alleviation.

CON-03 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/market/Market.sol (v1): 202, 209, 210, 214, 223; contracts/market/MarketRouter.sol (v1): 337, 338; contracts/market/MarketValid.sol (v1): 440; contracts/oracle/FastPriceFeed.sol (v1): 96; contracts/oracle/Price.sol (v1): 68, 72, 76; contracts/vault/RewardDistributor.sol (v1): 30, 31; contracts/vault/VaultReward.sol (v1): 44	Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

CON-04 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/market/MarketRouter.sol (v1): 193; contracts/market/OrderMgr.sol (v1): 57; contracts/vault/VaultReward.sol (v1): 88, 90	Acknowledged

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

No alleviation.

CON-07 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Logical Issue	Minor	contracts/fee/FeeRouter.sol (v1): 111, 125, 126~131, 132~137; contracts/market/PositionSubMgr.sol (v1): 302	Resolved

Description

Type casting refers to changing an variable of one data type into another. The code contains an unsafe cast between integer types, which may result in unexpected truncation or sign flipping of the value.

```
111             uint256(_fees),
```

- The type conversion `uint256(_fees)` from type int256 to type uint256 may flip the value's sign.

```
125         uint256 _fees = uint256(IFeeVault(feeVault).accountFees(account));
```

- The type conversion `uint256(IFeeVault(feeVault).accountFees(account))` from type int256 to type uint256 may flip the value's sign.

```
126         uint256 _buyFee = uint256(
127             IFeeVault(feeVault).accountKindFees(
128                 account,
129                 uint8(FeeType.BuyLpFee)
130             )
131         );
```

- The type conversion `uint256(IFeeVault(feeVault).accountKindFees(account,uint8(FeeType.BuyLpFee)))` from type int256 to type uint256 may flip the value's sign.

```
132         uint256 _sellFee = uint256(
133             IFeeVault(feeVault).accountKindFees(
134                 account,
135                 uint8(FeeType.SellLpFee)
136             )
137         );
```

- The type conversion `uint256(IFeeVault(feeVault).accountKindFees(account,uint8(FeeType.SellLpFee)))` from type int256 to type uint256 may flip the value's sign.

```
302                     uint256(_position.collateral.toInt256() - fees)
```

- The type conversion `uint256(_position.collateral.toInt256() - fees)` from type int256 to type uint256 may flip the value's sign.

Recommendation

It is recommended to check the bounds of integer values before casting. Alternatively, consider using the `SafeCast` library from OpenZeppelin to perform safe type casting and prevent undesired behavior.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/cf86fd9962701396457e50ab0d6cc78aa29a5ebc/contracts/utils/math/SafeCast.sol

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

CPF-02 | INCONSISTENT PRICE PRECISION

Category	Severity	Location	Status
Inconsistency	Minor	contracts/oracle/ChainPriceFeed.sol (v1): 42, 91	Acknowledged

Description

The price returned by `getLatestPrice()` uses price feed original precision, while the price returned by `getPrice()` uses fixed 30 decimals. It may cause confusion and incorrect calculation.

Recommendation

Recommend making sure price precisions are consistent.

Alleviation

The client confirmed that the current price precision implementation aligns with the original project design.

CVB-08 "VERIFYOUTASSETS" ALWAYS RETURN "TRUE"

Category	Severity	Location	Status
Volatile Code	Minor	contracts/vault/CoreVault.sol (v1): 154~159	Resolved

Description

The "verifyOutAssets" function contains no logic and always returns true. This function is called in the "transferFromVault" function to perform some sort of verification.

```
154     function verifyOutAssets(
155         address /* to */,
156         uint256 /* amount */
157     ) external pure override returns (bool) {
158         return true;
159     }
```

Recommendation

It is recommended to include logic in the function or remove the function if it is unnecessary.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

FFB-02 | UNUSED VALUE

Category	Severity	Location	Status
Inconsistency	Minor	contracts/fee/FundFee.sol (v1): 60~62	Resolved

Description

The smart contract contains one or more values that are not used, which can lead to unnecessary complexity and reduced maintainability.

```
60              interval =
61                  (intervals[i] / MIN_FUNDING_INTERVAL) *
62                  MIN_FUNDING_INTERVAL;
```

- The variable `interval` is never used after this assignment.

Recommendation

It is advised to ensure that all necessary values are used, and remove redundant values.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

FRB-02 | INCONSISTENT FEE VALUE

Category	Severity	Location	Status
Inconsistency	Minor	fee/FeeRouter.sol (v2): 138~139	Acknowledged

Description

```
133         uint256 _amount = IERC20(token).allowance(msg.sender, address(this));
134         if (_amount == 0) {
135             return;
136         }
137
138         IERC20(token).safeTransferFrom(msg.sender, feeVault, _amount);
139         IFeeVault(feeVault).increaseFees(msg.sender, account, fees);
```

The collected fee `amount` does not match the value of `fees` if collateral cannot cover PnL. Consequently, the fee information stored in the `FeeVault` will be inaccurate.

Recommendation

We recommend tracking the accurate fee information.

Alleviation

[Blex Team]: Issue acknowledged. I won't make any changes for the current version.

MAK-02 USER INPUT ORACLE PRICE WILL BE USED IN THE POSITION CALLBACK IF `sizeDelta` IS ZERO

Category	Severity	Location	Status
Volatile Code	Minor	contracts/market/PositionAddMgr.sol (v1): 56~57, 196; contracts/market/PositionSubMgr.sol (v1): 49~50, 217	Resolved

Description

The user input `_oraclePrice` of `MarketDataTypes.UpdatePositionInputs` will be updated by the oracle feed only when the `sizeDelta` is not zero. If the logic of a market plugin depends on the oracle price, it has the potential to be manipulated by a malicious user.

Recommendation

Consider using the price from the price feed to mitigate potential risk.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MAK-03 | INCONSISTENT COMMENTS AND VALIDATION LOGIC

Category	Severity	Location	Status
Inconsistency	Minor	contracts/market/MarketValid.sol (v1): 191; contracts/market/interfaces/IMarketValid.sol (v1): 112, 115	Resolved

Description

The comment for `maxLeverage` in `IMarketValid` states "0为不限制". However, the `validPay()` function does not handle this scenario.

The comment for `maxTradeAmount` in `IMarketValid` states "0为不限制". However, the validation logic does not allow unlimited leverage.

Recommendation

We recommend removing the aforementioned comments.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MAK-04 | MINIMUM SLIPPAGE PRICE IS NOT USED

Category	Severity	Location	Status
Inconsistency	Minor	contracts/market/MarketConfigStruct.sol (v1): 34~48; contracts/market/MarketValid.sol (v1): 273	Acknowledged

Description

The `validSlippagePrice()` function is missing a check to ensure that the input slippage is not lower than the minimum slippage price.

Recommendation

Consider adding the minimum slippage price validation or removing the minimum slippage configuration.

Alleviation

No alleviation.

MAK-05 | OUT OF SCOPE DEPENDENCIES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/market/Market.sol (v1): 54, 70; contracts/market/MarketStorage.sol (v1): 42	Acknowledged

Description

The `ROLE_POS_KEEPER` is responsible for the following tasks:

1. Monitor the leverage of positions and liquidate them if they exceed the maximum leverage threshold.
2. Monitor the listed orders and their trigger prices, and execute the triggered orders to increase or decrease positions.

The `ROLE_POS_KEEPER` operates off-chain and is not within the scope of this audit. The audit treats the `ROLE_POS_KEEPER` as a black box and assumes its functional correctness. However, if the `ROLE_POS_KEEPER` is compromised and fails to liquidate positions or execute orders in a timely manner, it will result in losses for the market and users.

The BLEX protocol supports `plugins` to be executed after position/order changes. The audit treats the `plugins` as a black box and assumes their functional correctness. If `plugins` execution fails, it will result in unexpected issues for the market.

Recommendation

We encourage the team to constantly monitor the status of `ROLE_POS_KEEPER` and ensure its security and functionality correctness.

Alleviation

No alleviation.

MVB-02 | INCORRECT INPUT FOR `validSize()` FUNCTION

Category	Severity	Location	Status
Inconsistency	Minor	contracts/market/MarketValid.sol (v1): 174	Resolved

Description

The input `_vars._order.price` is being incorrectly used for the parameter `_sizeDelta` of `validSize()` function.

```
174         validSize(0, _vars._order.price, true);
```

```
179     function validSize(
180         uint256 _size,
181         uint256 _sizeDelta,
182         bool _isIncrease
183     ) public pure override {
```

Recommendation

We recommend using `_vars._order.size` instead, or removing `validSize()` function call as it does not perform any check.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MVB-03 | INCORRECT ERROR MESSAGE

Category	Severity	Location	Status
Inconsistency	Minor	contracts/market/MarketValid.sol (v1): 210	Resolved

Description

The error message provided does not accurately describe the error.

```
210         require(conf.getDecrOrderLmt() >= decrOrderCount + 1, "trigger>10");
```

Recommendation

Consider updating the message to reflect the check being performed.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

OMB-03 COLLATERAL TO BE REFUNDED SHOULD SUBTRACT THE `execFee`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/market/OrderMgr.sol (v1): 205	Resolved

Description

During the order cancellation process, when `isExec` and `_isIncrease` are true, and `_isTransferToUser` is false, an `execFee` will be charged, which should be subtracted from the collateral to be refunded.

Recommendation

Consider subtracting the `execFee` from the `collateralRefund`.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PRI-02 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/oracle/Price.sol (v1): 27~28	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to manipulated prices. In addition, upgrades of third parties can possibly create severe impacts, such as manipulated prices, etc.

```
27      address public chainPriceFeed;
28      address public gmxPriceFeed;
```

- The contract `Price` interacts with third party contract with `IChainPriceFeed` interface via `chainPriceFeed`.
- The contract `Price` interacts with third party contract with `IGmxPriceFeed` interface via `gmxPriceFeed`.

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

No alleviation.

PSB-02 LACK OF INPUT VALIDATION ON `account` FOR FUNCTION `remove()`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/position/PositionStore.sol (v1): 76~78	Resolved

Description

If the `account` does not exist in `positionKeys` set, the function `remove()` should revert with error message.

Recommendation

Recommend adding a check to make sure `account` does exist in `positionKeys` set.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-02 `transToUser` SHOULD BE ZERO IF THE COLLATERAL IS INSUFFICIENT TO COVER FEES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/market/PositionSubMgr.sol (v1): 267, 299	Resolved

Description

```
263             if (
264                 (_params.liqState > 0 || isCloseAll) &&
265                 fees > _position.collateral.toInt256()
266             ) {
267                 transToUser += _position.collateral.toInt256();
```

No tokens will be transferred to the user if the collateral of the position is smaller than the fees (or fees plus loss). Therefore, the value of `transToUser` should be set to zero in this case.

Recommendation

We recommend setting the `transToUser` to zero in aforementioned scenarios.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-05 | POTENTIALLY INSUFFICIENT FUNDS TO TAKE PROFITS

Category	Severity	Location	Status
Volatile Code, Design Issue	Minor	contracts/market/PositionSubMgr.sol (v1): 287	Acknowledged

Description

When a position earns profits, the market withdraws tokens from the vault to cover those profits. However, if the profit amount is excessively large and the token balance in the vault is insufficient, the transaction will be reverted.

Recommendation

We recommend configuring the market properly by setting reasonable values for the market net size, maximum leverage, and maximum trade amount.

Alleviation

No alleviation.

PSM-07 | LACK OF SANITY CHECK FOR `liqState`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/market/PositionSubMgr.sol (v1): 210	Resolved

Description

The process of decreasing a position validates liquidation only when `liqState == 0` and `_params._sizeDelta != _position.size`.

```
210         if (_params.liqState == 0 && _params._sizeDelta != _position.size)
211             validLiq(_params._account, _params._isLong);
```

However, the `decreasePosition()` function is missing a check to ensure that `_params.liqState` is a valid value. As a result, the input `_params.liqState` can be set to an invalid value, such as 3, to bypass the liquidation validation.

Recommendation

We recommend adding a check to ensure the input `_params.liqState` is valid and calling `validLiq()` if `_params._sizeDelta != _position.size`.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-08 COLLATERAL SHOULD BE DECREASED TO ZERO IF COLLATERAL IS INSUFFICIENT TO COVER FEES

Category	Severity	Location	Status
Volatile Code	Minor	contracts/market/PositionSubMgr.sol (v1): 263~266, 295~297	Resolved

Description

if `fees > _position.collateral.toInt256()`, it indicates the collateral of the position is insufficient to cover fees. In this case, regardless the `_params.liqState` or `isCloseAll` status, the new collateral `newCollateralUnsigned` should be zero.

Recommendation

We recommend updating the condition for aforementioned scenario.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-09 CLOSE THE ENTIRE POSITION YIELDS A `PartialLiquidation` REASON

Category	Severity	Location	Status
Inconsistency	Minor	contracts/market/PositionSubMgr.sol (v1): 119, 123	Resolved

Description

If the decrease in size of a position, represented by `_params._sizeDelta`, is equal to the current size of the position (`_position.size`), the decreasing position process is considered to be 'isCloseAll'. In this scenario, if `_params.liqState` is equal to 2, the order deletion reason is set as `CancelReason.PartialLiquidation`. The reason for order deletion does not align with the actual operation being performed.

Recommendation

Consider changing to a reasonable cancellation reason instead.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

PSM-11 | COLLECTS EXECUTION FEE WHEN REMOVING ORDERS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/market/PositionSubMgr.sol (v1): 136	Acknowledged

Description

If users close a position and `isCloseAll` is true, the associated close orders of the account will be removed. These removed orders are not executed, but the variable `_params.execNum` will track the removed order number. Subsequently, the market will collect execution fees.

Recommendation

Consider not charging execution fees for the unexecuted orders.

Alleviation

[Blex Team]: Issue acknowledged. I won't make any changes for the current version.

VRU-02 COMMENT ABOUT `sell()` PARAMETER `amount` DOES NOT MATCH ACTUAL CODE

Category	Severity	Location	Status
Coding Issue	Minor	contracts/vault/VaultRouter.sol (v1): 78, 85, 91	Acknowledged

Description

The comment says "@param amount The amount of assets to be sold from the vault." but the actual code calls vault `redeem()` function which takes vault shares as parameter, not amount of assets.

Recommendation

Recommend reviewing the code and comment to make sure they are consistent.

Alleviation

From BLEX team: "Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement."

CVB-01 POTENTIAL INCORRECT `totalAssets()` IMPLEMENTATION

Category	Severity	Location	Status
Coding Issue	Informational	contracts/vault/CoreVault.sol (v1): 71	Acknowledged

Description

The `vaultRouter.getAUM()` method includes unrealized market gains and losses when calculating the total assets. We understand the current implementation requirements. If some vault investors see significant market losses that have not been withdrawn, they will withdraw their assets from the vault in advance to avoid losses, leaving the losses to those who haven't withdrawn their assets. On the other hand, arbitrageurs can take advantage of the situation by depositing assets into the vault in advance if they detect significant unrealized market profits.

However, based on ERC4626, the `totalAssets()` should return the balance of underlying token in the vault, which is definitely different from `vaultRouter.getAUM()`. An incorrect `totalAssets()` will lead to incorrect calculation in `_convertToShares()/_convertToAssets()/_isVaultCollateralized()` such that tokens may be stolen from the vault. See https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.3/contracts/token/ERC20/extensions/ERC4626.sol

Recommendation

It is recommended that the team reviews the implementation and ensures that the "totalAssets" function is intended to return "vaultRouter.getAUM();" instead of the balance of the underlying token in the vault.

Alleviation

[Blex Team]: As stated in the description, it is necessary to add unrealized gains and losses to the total assets in order to solve the described problem. When the total assets do not include unrealized gains and losses, the closing action of market traders will lead to an increase or decrease in the total assets of the CoreVault, but after the unrealized gains and losses are added to the total assets, the closing action of market traders will not lead to a change in the total assets of the CoreVault. Because the closing action of the market trader will lead to the change of the profit and loss of the position and the transfer in or transfer out of the CoreVault funds, the profit will inevitably lead to the transfer out of the CoreVault, and the loss will inevitably lead to the transfer in of the CoreVault, so the total assets of the CoreVault will not change when the two occur at the same time in a transaction.

FFB-03 | TYPO OF CONSTANT NAME

Category	Severity	Location	Status
Coding Style	Informational	contracts/fee/FundFee.sol (v1): 14, 218	Resolved

Description

`DEFAILT_RATE_DIVISOR` should be `DEFAULT_RATE_DIVISOR` .

Recommendation

Recommend correcting the constant name.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

FVB-01 | `decreaseFees()` ADDS VALUES ON CURRENT FEES

Category	Severity	Location	Status
Incorrect Calculation	Informational	contracts/fee/FeeVault.sol (v1): 81~82	Acknowledged

Description

Why does `decreaseFees()` add value on current fees?

Recommendation

We would like to confirm if it is intended.

Alleviation

[Blex Team]: Because the value of the parameter fees is negative.

MRB-01 | MARKETS NEED TO BE CREATED BY THE `MarketFactory`

Category	Severity	Location	Status
Volatile Code	Informational	contracts/market/MarketReader.sol (v1): 34	Acknowledged

Description

The `MarketReader` contract retrieves market details from the `markets` within the `MarketFactory` contract. However, if any market is not created by the `MarketFactory` contract, the market data obtained from the `MarketReader` would be inaccurate.

Recommendation

We recommend creating every market with the `MarketFactory` contract.

Alleviation

No alleviation.

MSB-01 AVOID POTENTIAL STORAGE CONFLICTS IN MARKET UPGRADES

Category	Severity	Location	Status
Volatile Code	Informational	contracts/market/MarketStorage.sol (v1): 12	Acknowledged

Description

The `Market` contract and the implementation contracts for market logic, namely `OrderMgr`, `PositionAddMgr`, and `PositionSubMgr`, share the same storage. If the implementation contract upgrades involve storage changes, it can potentially lead to storage conflicts, which in turn may result in unexpected security issues.

Recommendation

We recommend ensuring market state variables are not changed and appending new variables if required.

Alleviation

No alleviation.

MVB-01 | DISCUSSION ON TAKE PROFIT AND STOP LOSS PRICES

Category	Severity	Location	Status
Volatile Code	Informational	contracts/market/MarketValid.sol (v1): 150, 157	Acknowledged

Description

The take-profit price must be higher than the trigger price for a long order, while the stop-loss price must be lower than the trigger price for a short order.

Recommendation

Is it possible for the take-profit price to fall within a reasonable price range? For instance, if the trigger price for a close order is set to 10 ether, and the mark price reaches 10 ether and subsequently drops to 9.99 ether within a short period of time, users may want to close their position even if the price is slightly lower than the trigger price.

Alleviation

[Blex Team]: The `feeReceiver` parameter is added to the function related to the execution process to encourage the third party to participate in the execution process.

OPTIMIZATIONS | BLEX

ID	Title	Category	Severity	Status
CON-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
MRU-01	Transfer Collateral Tokens If `collateralDelta` Is Not Zero	Gas Optimization	Optimization	Acknowledged
PSM-03	Query The Price Twice Within A Function	Gas Optimization	Optimization	Resolved
RDB-01	Unused `ReentrancyGuard`	Gas Optimization	Optimization	Acknowledged
RDB-02	Unnecessary Usage Of `SafeMath`	Gas Optimization	Optimization	Acknowledged

CON-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/fee/FundFee.sol (v1): 9; contracts/market/MarketValid.sol (v1): 437; contracts/position/PositionStore.sol (v1): 18	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

MRU-01 TRANSFER COLLATERAL TOKENS IF `collateralDelta` IS NOT ZERO

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/market/MarketRouter.sol (v1): 163, 193	Acknowledged

Description

The `increasePosition()` method is only needed to transfer the user's collateral tokens to the market if the user wants to add collateral.

Recommendation

Consider transferring collateral tokens from `msg.sender` when `_inputs.collateralDelta` is not zero.

Alleviation

No alleviation.

PSM-03 | QUERY THE PRICE TWICE WITHIN A FUNCTION

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/market/PositionSubMgr.sol (v1): 104	Resolved

Description

In the `_liquidatePosition()` function, the price is fetched and stored in the variable `_vars._oraclePrice`. This allows the price value to be reused rather than having to query the price again.

Recommendation

Consider using the stored price value when calling `_valid().isLiquidate()`.

Alleviation

Fixed in commit 9dafc868e4584914283c788a1ea7ab62bc00fe1f.

RDB-01 | UNUSED `ReentrancyGuard`

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/vault/RewardDistributor.sol (v1): 12	Acknowledged

Description

The contract `RewardDistributor` extends `ReentrancyGuard`, but the modifier `nonReentrant()` is never used.

Recommendation

Recommend removing `ReentrancyGuard` if it is not intended to be used.

Alleviation

From BLEX team: "Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement."

RDB-02 | UNNECESSARY USAGE OF `SafeMath`

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/vault/RewardDistributor.sol (v1): 13	Acknowledged

Description

Starting from version 0.8.0, Solidity builtin arithmetic operations will revert on underflow/overflow. So `SafeMath` is no longer needed.

Recommendation

Recommend removing usage of `SafeMath`.

Alleviation

From BLEX team: "Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement."

FORMAL VERIFICATION | BLEX

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-never-return-false	`transfer` Never Returns `false`

Property Name	Title
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State

Property Name	Title
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-never-return-false	`approve` Never Returns `false`

Verification Results

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
  - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
  - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.
- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
  - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract CoreVault (contracts/vault/CoreVault.sol) In Commit 14d6a91b34d778c9d50d17d18ca4d19133fe68ea

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-change-state	True	
erc20-transfer-false	True	
erc20-transfer-recipient-overflow	False	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	False	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	True	
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-false	True	
erc20-approve-change-state	True	
erc20-approve-never-return-false	True	

APPENDIX | BLEX

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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.

Details on Formal Verification

Technical description

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The

model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to an ERC-20 token contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property definitions

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time steps. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`.

In the following, we list those property specifications.

Properties for ERC-20 function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address.

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
[](started(contract.transfer(to, value), to == address(0))
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers.

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to != msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[to] + value <= type(uint256).max && _balances[to] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers.

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to == msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[msg.sender] >= 0
          && _balances[msg.sender] <= type(uint256).max
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender
   && _balances[to] >= 0 && value >= 0
   && _balances[to] + value <= type(uint256).max
   && _balances[msg.sender] >= 0 && _balances[msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transfer(to, value), return
             ==> _balances[msg.sender] == old(_balances[msg.sender]) - value
                 && _balances[to] == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`.

Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender
  && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
     ==> <>(finished(contract.transfer(to, value), return
         ==> _balances[to] == old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to)
  ==> <>(finished(contract.transfer(to, value), return
      ==> (_totalSupply == old(_totalSupply) && _allowances == old(_allowances)
           && _balances[p1] == old(_balances[p1])  ))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance.

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender]
   && _balances[msg.sender] >= 0 && value <= type(uint256).max)
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance.

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
[](started(contract.transfer(to, value), to != msg.sender
   && _balances[to] + value > type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max
   && _balances[msg.sender] <= type(uint256).max
   && value > 0 && value <= _balances[msg.sender])
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return) || finished(contract.transfer(to, value), _balances[to]
                      > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transfer-false

If Function `transfer` Returns `false` , the Contract State Has Not Been Changed.

If the `transfer` function in contract `contract` fails by returning `false` , it must undo all state changes it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return]
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transfer-never-return-false

Function `transfe` Never Returns `false`.

The transfer function must never return `false` to signal a failure.

Specification:

```
[](!(finished(contract.transfer, !return)))
```

Properties for ERC-20 function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && to != address(0) && from != to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && _balances[to] + value <= type(uint256).max
   && value >= 0 && _balances[to] >= 0 && _balances[from] >= 0
   && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers.

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && from == to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && value >= 0 && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] <= type(uint256).max)
     ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers.

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0
&& _balances[from] >= 0 && _balances[from] <= type(uint256).max
&& _balances[to] >= 0 && _balances[to] + value <= type(uint256).max)
   ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> _balances[from] == old(_balances[from]) - value
              && _balances[to] == old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to
   && value >= 0 && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
             ==> _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0
   && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max && _balances[to] >= 0
   && _balances[to] <= type(uint256).max && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> ((_allowances[from][msg.sender]
                == old(_allowances[from][msg.sender]) - value)
               || (_allowances[from][msg.sender]
                   == old(_allowances[from][msg.sender])
                      && (from == msg.sender
                          || old(_allowances[from][msg.sender])
                             == type(uint256).max))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to
   && (p2 != from || p3 != msg.sender))
    ==> <>(finished(contract.transferFrom(from, to, amount), return
     ==> (_totalSupply == old(_totalSupply) && _balances[p1] == old(_balances[p1])
         && _allowances[p2][p3] == old(_allowances[p2][p3])  ))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from]
   && _balances[from] >= 0 && _balances[from] <= type(uint256).max)
       ==> <>(reverted(contract.transferFrom)
              || finished(contract.transferFrom, !return)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
  [](started(contract.transferFrom(from, to, value), value > _allowances[from]
[msg.sender]
    && _allowances[from][msg.sender] >= 0 && value <= type(uint256).max)
    ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), return
             && (msg.sender == from
                 || _allowances[from][msg.sender] == type(uint256).max))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance.

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != to
   && _balances[to] + value > type(uint256).max && value <= type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
      ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), _balances[to]
              > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed.

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`.

The `transferFrom` function must never return `false`.

Specification:

```
[](!(finished(contract.transferFrom, !return)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds.

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable.

The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, return == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State.

The `totalSupply` function in contract contract must not change any state variables.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, _totalSupply == old(_totalSupply)
          && _balances == old(_balances) && _allowances == old(_allowances)  )))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds.

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value.

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State.

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), _totalSupply == old(_totalSupply)
          && _balances == old(_balances)
          && _allowances == old(_allowances)  )))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds.

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value.

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
           return == _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State.

Function `allowance` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
   _totalSupply == old(_totalSupply) && _balances == old(_balances)
               && _allowances == old(_allowances)  )))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address.

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
[](started(contract.approve(spender, value), spender == address(0))
   ==> <>(reverted(contract.approve)
          || finished(contract.approve(spender, value), !return)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs.

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
[](started(contract.approve(spender, value), spender != address(0))
   ==> <>(finished(contract.approve(spender, value), return)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly.

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && value >= 0 && value <= type(uint256).max)
       ==> <>(finished(contract.approve(spender, value), return
              ==> _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes.

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && (p1 != msg.sender || p2 != spender))
       ==> <>(finished(contract.approve(spender, value), return
              ==> _totalSupply == old(_totalSupply) && _balances == old(_balances)
                  && _allowances[p1][p2] == old(_allowances[p1][p2])  )))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed.

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.approve(spender, value))
   ==> <>(finished(contract.approve(spender, value), !return
          ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
               && _allowances == old(_allowances)  ))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false`.

The function `approve` must never returns `false`.

Specification:

```
 [](!(finished(contract.approve, !return)))
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

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 considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 CertiK is subject to dependencies and 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.