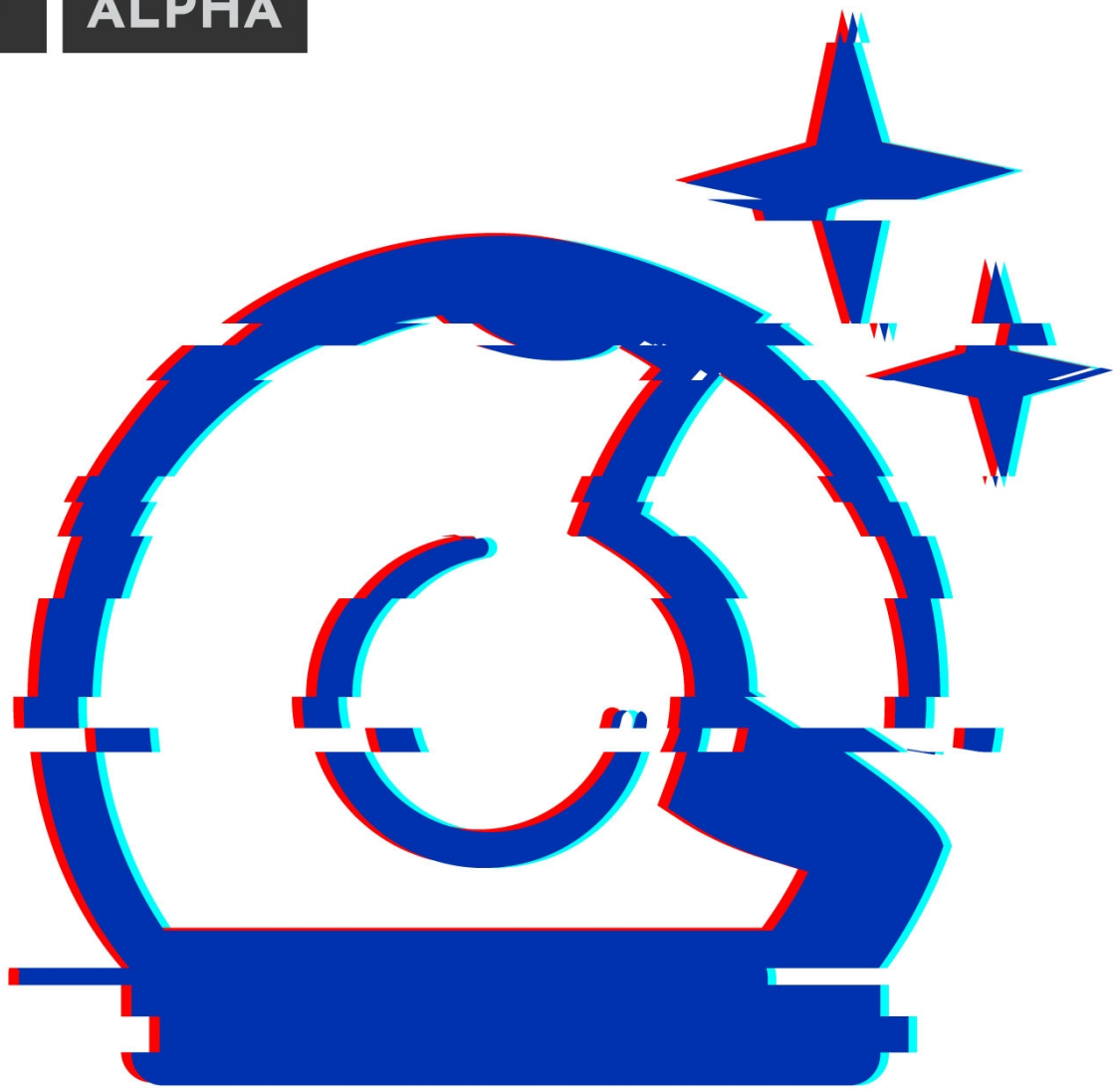


PROJECT

Gamma Protocol

CLIENT

Opyn

DATE

August 2021

REVIEWERS

Daniel Luca

@cleanunicorn

Andrei Simion

@andreiashu

Table of Contents

Details

- **Client** Opyn
- **Date** August 2021
- **Lead reviewer** Daniel Luca (@cleanunicorn)
- **Reviewers** Daniel Luca (@cleanunicorn), Andrei Simion (@andreiashu)
- **Repository**: Gamma Protocol
- **Commit hash** `67a2bff57ec49c4bb7c9c454c8ad945fd5bdcf51`
- **Technologies**
  - Solidity
  - Node.JS

Issues Summary

SEVERITY	OPEN	CLOSED
Informational	2	0
Minor	2	0
Medium	3	0
Major	0	0

Executive summary

This report represents the results of the engagement with **Opyn** to review **Gamma Protocol**.

The review was conducted over the course of **2 weeks** from **August 16 to August 27, 2021**. A total of **20 person-days** were spent reviewing the code.

Week 1

During the first week, we started to get familiarized with the code and the project. We reviewed the code from the beginning to the end of the week.

We set up a few meetings throughout the week to discuss the code and learn how to navigate the codebase. We also discussed the project goals and the project scope.

We identified a few minor issues that we presented to the development team. A few of the already implemented and deployed features needed more clarification in terms of how they are currently configured, and how they fit into the overall project.

Week 2

We continued to keep communication open with the development team while navigating the code and trying out different attack vectors.

Even though the project has a lot of code, the way it is structured makes it easy to navigate and understand. Some of its parts exist outside of the blockchain context, meaning that they push trusted data into the chain, which then is used to calculate different security parameters.

During the final days of the week we finalized the report and presented it to the client.

Scope

The initial review focused on the Gamma Protocol repository, identified by the commit hash `67a2bff57ec49c4bb7c9c454c8ad945fd5bdcf51`.

We focused on manually reviewing the codebase, searching for security issues such as, but not limited to, re-entrancy problems, transaction ordering, block timestamp dependency, exception handling, call stack depth limitation, integer overflow/underflow, self-destructible contracts, unsecured balance, use of origin, costly gas patterns, architectural problems, code readability.

Includes:

- core/AddressBook.sol
- core/Controller.sol
- core/MarginCalculator.sol
- core/MarginPool.sol
- core/Otoken.sol
- external/callees/PermitCallee.sol
- libs/MarginVault.sol
- other contracts supporting these files

Excludes:

- Everything else

Issues

Donate can lock funds in certain situations

Status Open Severity Medium

Description

`donate` function seems to allow users to donate an `_amount` of tokens of `_asset` to the margin pool:

code/contracts/core/Controller.sol#L316-L322

```
     * @notice send asset amount to margin pool
     * @dev use donate() instead of direct transfer() to store the balance in assetBalance
     * @param _asset asset address
     * @param _amount amount to donate to pool
     */
    function donate(address _asset, uint256 _amount) external {
        pool.transferToPool(_asset, msg.sender, _amount);
```

`transferToPool` is used in order to transfer the tokens from the user to the pool. This function takes care of the internal accounting inside the `MarginPool` contract and updates the `assetBalance` accordingly:

code/contracts/core/MarginPool.sol#L74-L83

```
    function transferToPool(
        address _asset,
        address _user,
        uint256 _amount
    ) public onlyController {
        require(_amount > 0, "MarginPool: transferToPool amount is equal to 0");
        assetBalance[_asset] = assetBalance[_asset].add(_amount);

        // transfer _asset _amount from _user to pool
        ERC20Interface(_asset).safeTransferFrom(_user, address(this), _amount);
```

We believe that the `farm` function was intended to be the one that would allow a designated farm account to withdraw the funds that were donated into the pool (ie. the difference between what is in the contract and `assetBalance`):

code/contracts/core/MarginPool.sol#L173-L178

```
        uint256 externalBalance = ERC20Interface(_asset).balanceOf(address(this));
        uint256 storedBalance = assetBalance[_asset];

        require(_amount <= externalBalance.sub(storedBalance), "MarginPool: amount to farm excee

        ERC20Interface(_asset).safeTransfer(_receiver, _amount);
```

However, the issue is that because `transferToPool` also updates the internal accounting in the `assetBalance` mapping, the `require` at line L176 in `MarginPool` contract will fail for any value of `_amount` higher than 0. Because the difference between the funds in the contract (`externalBalance`) and the amount accounted in the `assetBalance` mapping (`storedBalance`) will be 0 (*). This leads to the situation where the donated funds are stuck in the contract:

code/contracts/core/MarginPool.sol#L176

```
        require(_amount <= externalBalance.sub(storedBalance), "MarginPool: amount to farm excee
```

(*) this is true as long as users follow the recommendation in the `donate` documentation and only send transfers via this function. Any tokens sent *directly* to the `MarginPool` by use of `transfer` can actually be *withdrawn* / *farmed*.

Recommendation

Since this is a live contract and other systems might already be aware of the external `donate` function, it might make sense to just do a `ERC20Interface(_asset).safeTransfer` call directly, without calling the `transferToPool`. This way, when the `farm` function is called, the `ERC20Interface(_asset).balanceOf` call in `MarginPool` contract will indeed be higher than the `assetBalance[_asset]` and the difference can be farmed.

After further discussion with the development team, we understood the purpose of the `donate` method was to add funds to the protocol when it is under collateralized to cover for the missing funds. Anyone can call this method because they plan to have an emergency plan to add funds in the vault.

`donate` should make sure the assets handled are part of the whitelist

Description

Any actor can call `Controller.donate()` to push some ERC20 tokens into the pool.

code/contracts/core/Controller.sol#L315-L325

```
    /**
     * @notice send asset amount to margin pool
     * @dev use donate() instead of direct transfer() to store the balance in assetBalance
     * @param _asset asset address
     * @param _amount amount to donate to pool
     */
    function donate(address _asset, uint256 _amount) external {
        pool.transferToPool(_asset, msg.sender, _amount);

        emit Donated(msg.sender, _asset, _amount);
    }
```

There are other instances where tokens are pushed into the pool, but an additional whitelist validation is done in each case.

_depositLong

code/contracts/core/Controller.sol#L726-L731

```
    /**
     * @notice deposit a long oToken into a vault
     * @dev only the account owner or operator can deposit a long oToken, cannot be called when
     * @param _args DepositArgs structure
```

```
     */
    function _depositLong(Actions.DepositArgs memory _args)
```

code/contracts/core/Controller.sol#L740

```
        require(whitelist.isWhitelistedOtoken(_args.asset), "C17");
```

code/contracts/core/Controller.sol#L748

```
        pool.transferToPool(_args.asset, _args.from, _args.amount);
```

_depositCollateral

code/contracts/core/Controller.sol#L776-L781

```
    /**
     * @notice deposit a collateral asset into a vault
     * @dev only the account owner or operator can deposit collateral, cannot be called when sys
     * @param _args DepositArgs structure
     */
    function _depositCollateral(Actions.DepositArgs memory _args)
```

code/contracts/core/Controller.sol#L790

```
        require(whitelist.isWhitelistedCollateral(_args.asset), "C21");
```

code/contracts/core/Controller.sol#L802

```
        pool.transferToPool(_args.asset, _args.from, _args.amount);
```

In order to keep the same rigorous accounting of only whitelisted tokens being accounted for in the pool, a whitelist check should be done in `donate` before transferring tokens.

Recommendation

Add an additional `require` check to validate the transferred token to the pool in the method `donate`.

A user calling `sync` can be front-run

Status Open Severity Medium

Description

A user can call `Controller.sync()` to update their vault's latest timestamp.

code/contracts/core/Controller.sol#L439-L449

```
    /**
     * @notice sync vault latest update timestamp
     * @dev anyone can update the latest time the vault was touched by calling this function
     * vaultLatestUpdate will sync if the vault is well collateralized
     * @param _owner vault owner address
     * @param _vaultId vault id
     */
    function sync(address _owner, uint256 _vaultId) external nonReentrant notFullyPaused {
        _verifyFinalState(_owner, _vaultId);
        vaultLatestUpdate[_owner][_vaultId] = now;
    }
```

This timestamp is important when a vault is liquidated. The method `MarginCalculator.isLiquidatable` needs the vault's timestamp to be in the past as compared to the current timestamp.

code/contracts/core/MarginCalculator.sol#L445-L449

```
        // check that price timestamp is after latest timestamp the vault was updated at
        require(
            timestamp > _vaultLatestUpdate,
            "MarginCalculator: auction timestamp should be post vault latest update"
        );
```

If the vault was at some point in the past liquidatable, but now it isn't anymore, it could be liquidated, unless the user (or a benefactor) updates the user's vault by calling `Controller.sync(address _owner, uint256 _vaultId)`.

Also, if anyone is watching the blockchain and they see a transaction in the mempool calling `Controller.sync()` they can assume (and verify) if that vault could be liquidated. They could front-run the `Controller.sync()` transaction and liquidate the vault before the timestamp is updated.

The method seems to be vulnerable to front-running attacks.

Recommendation

Liquidating a user could check if the user can be liquidated at the time of the execution. This way, the user can't be front-run when trying to sync their vaults.

`notPartiallyPaused`, `notFullyPaused` and `onlyAuthorized` modifiers have unnecessary function counterparts

Status Open Severity Minor

Description

`notPartiallyPaused` modifier calls `_isNotPartiallyPaused` function in order to ensure that the state variable `systemPartiallyPaused` is not set to *true*:

code/contracts/core/Controller.sol#L218-L221

```
    modifier notPartiallyPaused {
        _isNotPartiallyPaused();

        _;
```

code/contracts/core/Controller.sol#L277-L279

```
    function _isNotPartiallyPaused() internal view {
        require(!systemPartiallyPaused, "C4");
    }
```

`_isNotPartiallyPaused` is an internal view function and is not called by any other function. This means that there is added gas costs but also mental energy expenditure for readers of the code and developers maintaining the contract.

The same logic applies to `notFullyPaused` and `onlyAuthorized` (they make use internal view functions which are called only from the modifier).

Recommendation

Include the `require` logic directly in the modifiers and delete the obsolete internal functions.

Opening a vault type defaults to type zero

Status Open Severity Minor

Description

A user can open a vault by calling `Controller.operate()` with a list of actions. One of those actions can be an open vault action.

code/contracts/core/Controller.sol#L426-L432

```
    /**
     * @notice execute a number of actions on specific vaults
     * @dev can only be called when the system is not fully paused
     * @param _actions array of actions arguments
     */
```

```
    function operate(Actions.ActionArgs[] memory _actions) external nonReentrant notFullyPaused
        (bool vaultUpdated, address vaultOwner, uint256 vaultId) = _runActions(_actions);
```

Each of those actions is executed in the controller, in the internal method `Controller._runActions`.

If the action type is equal to `Actions.ActionType.OpenVault`, a new vault is opened for the user.

code/contracts/core/Controller.sol#L665-L666

```
            if (actionType == Actions.ActionType.OpenVault) {
                _openVault(Actions._parseOpenVaultArgs(action));
```

But first the arguments needed are parsed by `Actions._parseOpenVaultArgs`.

code/contracts/libs/Actions.sol#L184-L189

```
    /**
     * @notice parses the passed in action arguments to get the arguments for an open vault acti
     * @param _args general action arguments structure
     * @return arguments for a open vault action
     */
    function _parseOpenVaultArgs(ActionArgs memory _args) internal pure returns (OpenVaultArgs m
```

The vault type is encoded in the `_args.data` property.

code/contracts/libs/Actions.sol#L196-L202

```
        if (_args.data.length == 32) {
            // decode vault type from _args.data
            vaultType = abi.decode(_args.data, (uint256));
        }

        // for now we only have 2 vault types
        require(vaultType < 2, "A3");
```

This value is checked to be valid (only type 0 and 1 accepted), but it defaults to 0. If the encoded sent data is invalid, the type defaults to 0. If the `_args.data.length` is not of size 32, the vault type defaults to 0.

Recommendation

It might help ensure the encoded data is valid by having a stricter validation of the input data.

A similar check is already done in `_parseLiquidateArgs`.

code/contracts/libs/Actions.sol#L323

```
        require(_args.data.length == 32, "A21");
```

External contract calls have unclear purpose

Status Open Severity Informational

Description

A user can call `operate` with a number of actions they want to execute in one transaction.

code/contracts/core/Controller.sol#L426-L437

```
    /**
     * @notice execute a number of actions on specific vaults

     * @dev can only be called when the system is not fully paused
     * @param _actions array of actions arguments
     */
    function operate(Actions.ActionArgs[] memory _actions) external nonReentrant notFullyPaused
        (bool vaultUpdated, address vaultOwner, uint256 vaultId) = _runActions(_actions);
        if (vaultUpdated) {
            _verifyFinalState(vaultOwner, vaultId);
            vaultLatestUpdate[vaultOwner][vaultId] = now;
        }
    }
```

These actions can be very specific to the system currently being reviewed, but they can also be more generic. The generic actions can be specified by sending an action of type `Actions.ActionType.Call`.

code/contracts/core/Controller.sol#L685-L687

```
            } else if (actionType == Actions.ActionType.Call) {
                _call(Actions._parseCallArgs(action));
            }
```

The method `_call` does an external call to a specified contract.

code/contracts/core/Controller.sol#L1037

```
        CalleeInterface(_args.callee).callFunction(msg.sender, _args.data);
```

This can be used as a callback function to trigger external code the user needs in between the vault-related actions.

There is also a modifier on top of this method that only allows a specific set of external contracts or any contract.

code/contracts/core/Controller.sol#L262-L272

```
    /**
     * @notice modifier to check if the called address is a whitelisted callee address
     * @param _callee called address
     */
    modifier onlyWhitelistedCallee(address _callee) {
        if (callRestricted) {
            require(_isCalleeWhitelisted(_callee), "C3");
        }

        _;
    }
```

This functionality is significantly restricted in a few ways:

- `Controller` contract calls only a specific method of the destination contract `.callFunction`
- list of arguments is fixed and already set up `(msg.sender, _args.data)`

The result of the execution is not processed in any way. Whether the call fails or succeeds, the execution is not affected.

Consider the following setup which makes an external call and emits different events based on the execution result.

```
contract External {
    bool public shouldFail;

    function setFail(bool _shouldFail) public {
        shouldFail = _shouldFail;
    }

    function maybeFail() public {
        if (shouldFail) {
            revert("Emitted error");
        }
    }
}

contract Caller {
    External public e;

    constructor () {
        e = new External();
    }
```

```
    function callMaybeFail() external {
        try e.maybeFail() {
            emit Success();
        } catch Error(string memory _reason) {
            emit ResultString(false, _reason);
        } catch (bytes memory _lowLevelReason) {
            emit ResultBytes(false, _lowLevelReason);
        }
    }

    event ResultBytes(bool, bytes);
    event ResultString(bool, string);
    event Success();
}
```

The example above lets the contract behave differently based on the success of the execution. It is unclear whether this additional functionality is needed or required in the current context. Additional complexity needs to be added in order to complete the functionality.

Extreme care needs to be taken if the external call functionality is extended.

In its current form, it seems like a very limited type of functionality, but an extension of this functionality easily brings major security problems.

A clearer, more specific description of the required functionality needs to be reviewed before implementing anything else in this direction.

Recommendation

Consider reviewing a more specific implementation plan before going forward with this functionality.

EIP-2612 permit is likely to change

Description

The contract `PermitCallee` uses the EIP-2612: permit – 712-signed approvals standard to update the allowance for a user, using a provided signature.

code/contracts/external/callees/PermitCallee.sol#L11-L31

```
/**
 * @title PermitCallee
 * @author Opyn Team
 * @dev Contract for executing permit signature
 */
```

```
contract PermitCallee is CalleeInterface {
    function callFunction(address payable _sender, bytes memory _data) external override {
        (
            address token,
            address owner,
            address spender,
            uint256 amount,
            uint256 deadline,
            uint8 v,
            bytes32 r,
            bytes32 s
        ) = abi.decode(_data, (address, address, address, uint256, uint256, uint8, bytes32, byte

        IERC20PermitUpgradeable(token).permit(owner, spender, amount, deadline, v, r, s);
    }
}
```

This standard isn't officially recommended anymore and will likely change in the future. This might be because of EIP-3074: AUTH and AUTHCALL opcodes.

Recommendation

Consider deprecating or not relying too much on the currently implemented functionality and consider using the more flexible **EIP-3074: AUTH and AUTHCALL opcodes**.

Artifacts

Surya

Sūrya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

📘 Controller

Files Description Table

File Name	SHA-1 Hash
core/Controller.sol	66a9209d597b4d4ff528d72730cc3e25af33badc

Contracts Description Table

Contract	Type	Bases	
└	**Function Name**	**Visibility**	**Mutability**
Controller	Implementation	Initializable, OwnableUpgradeSafe, ReentrancyGuardUpgradeSafe	
└	_isNotPartiallyPaused	Internal 🔒	
└	_isNotFullyPaused	Internal 🔒	
└	_isAuthorized	Internal 🔒	
└	initialize	External ❗	🛑
└	donate	External ❗	🛑
└	setSystemPartiallyPaused	External ❗	🛑
└	setSystemFullyPaused	External ❗	🛑
└	setFullPauser	External ❗	🛑
└	setPartialPauser	External ❗	🛑
└	setCallRestriction	External ❗	🛑
└	setOperator	External ❗	🛑
└	refreshConfiguration	External ❗	🛑
└	setNakedCap	External ❗	🛑
└	operate	External ❗	🛑
└	sync	External ❗	🛑
└	isOperator	External ❗	
└	getConfiguration	External ❗	
└	getProceed	External ❗	
└	isLiquidatable	External ❗	
└	getPayout	Public ❗	
└	isSettlementAllowed	External ❗	
└	canSettleAssets	External ❗	
└	getAccountVaultCounter	External ❗	

Contract	Type	Bases	
└	hasExpired	External ❗	
└	getVault	External ❗	
└	getVaultWithDetails	Public ❗	
└	getNakedCap	External ❗	
└	getNakedPoolBalance	External ❗	
└	_runActions	Internal 🔒	🛑
└	_verifyFinalState	Internal 🔒	
└	_openVault	Internal 🔒	🛑
└	_depositLong	Internal 🔒	🛑
└	_withdrawLong	Internal 🔒	🛑
└	_depositCollateral	Internal 🔒	🛑
└	_withdrawCollateral	Internal 🔒	🛑
└	_mintOtoken	Internal 🔒	🛑
└	_burnOtoken	Internal 🔒	🛑
└	_redeem	Internal 🔒	🛑
└	_settleVault	Internal 🔒	🛑
└	_liquidate	Internal 🔒	🛑
└	_call	Internal 🔒	🛑
└	_checkVaultId	Internal 🔒	
└	_isNotEmpty	Internal 🔒	
└	_isCalleeWhitelisted	Internal 🔒	
└	_isLiquidatable	Internal 🔒	
└	_getOtokenDetails	Internal 🔒	

Contract	Type	Bases	
└	_canSettleAssets	Internal 🔒	
└	_refreshConfigInternal	Internal 🔒	🛑

📘 MarginCalculator

Files Description Table

File Name	SHA-1 Hash
core/MarginCalculator.sol	3a4048d34b7a3cf47549e3e4d5b9712e23918f7f

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifi**
MarginCalculator	Implementation	Ownable		
└		Public ❗	🛑	NO
└	setCollateralDust	External ❗	🛑	onlyOv
└	setUpperBoundValues	External ❗	🛑	onlyOv
└	updateUpperBoundValue	External ❗	🛑	onlyOv
└	setSpotShock	External ❗	🛑	onlyOv
└	setOracleDeviation	External ❗	🛑	onlyOv
└	getCollateralDust	External ❗		NO
└	getTimesToExpiry	External ❗		NO
└	getMaxPrice	External ❗		NO
└	getSpotShock	External ❗		NO
└	getOracleDeviation	External ❗		NO
└	getNakedMarginRequired	External ❗		NO
└	getExpiredPayoutRate	External ❗		NO
└	isLiquidatable	External ❗		NO
└	getMarginRequired	External ❗		NO

Contract	Type	Bases		
└	getExcessCollateral	Public ❗		NO
└	_getExpiredCashValue	Internal 🔒		
└	_getMarginRequired	Internal 🔒		
└	_getNakedMarginRequired	Internal 🔒		
└	_findUpperBoundValue	Internal 🔒		
└	_getPutSpreadMarginRequired	Internal 🔒		
└	_getCallSpreadMarginRequired	Internal 🔒		
└	_convertAmountOnLivePrice	Internal 🔒		
└	_convertAmountOnExpiryPrice	Internal 🔒		
└	_getDebtPrice	Internal 🔒		
└	_getVaultDetails	Internal 🔒		
└	_getExpiredSpreadCashValue	Internal 🔒		
└	_isNotEmpty	Internal 🔒		
└	_checkIsValidVault	Internal 🔒		
└	_isMarginableLong	Internal 🔒		
└	_isMarginableCollateral	Internal 🔒		

Contract	Type	Bases		
└	_getProductHash	Internal 🔒		
└	_getCashValue	Internal 🔒		
└	_getOtokenDetails	Internal 🔒		

📘 PermitCallee

Files Description Table

File Name	SHA-1 Hash
external/callees/PermitCallee.sol	b98ff2a706037baaa339e0f86e309798092ad81e

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
PermitCallee	Implementation	CalleeInterface		
└	callFunction	External ❗	🛑	NO ❗

📘 MarginVault

Files Description Table

File Name	SHA-1 Hash
libs/MarginVault.sol	5f067b7b716b0645029104f6de80ba8dcd279418

Contracts Description Table

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
MarginVault	Library			
L	addShort	External ❗	🛑	NO ❗
L	removeShort	External ❗	🛑	NO ❗
L	addLong	External ❗	🛑	NO ❗
L	removeLong	External ❗	🛑	NO ❗
L	addCollateral	External ❗	🛑	NO ❗
L	removeCollateral	External ❗	🛑	NO ❗

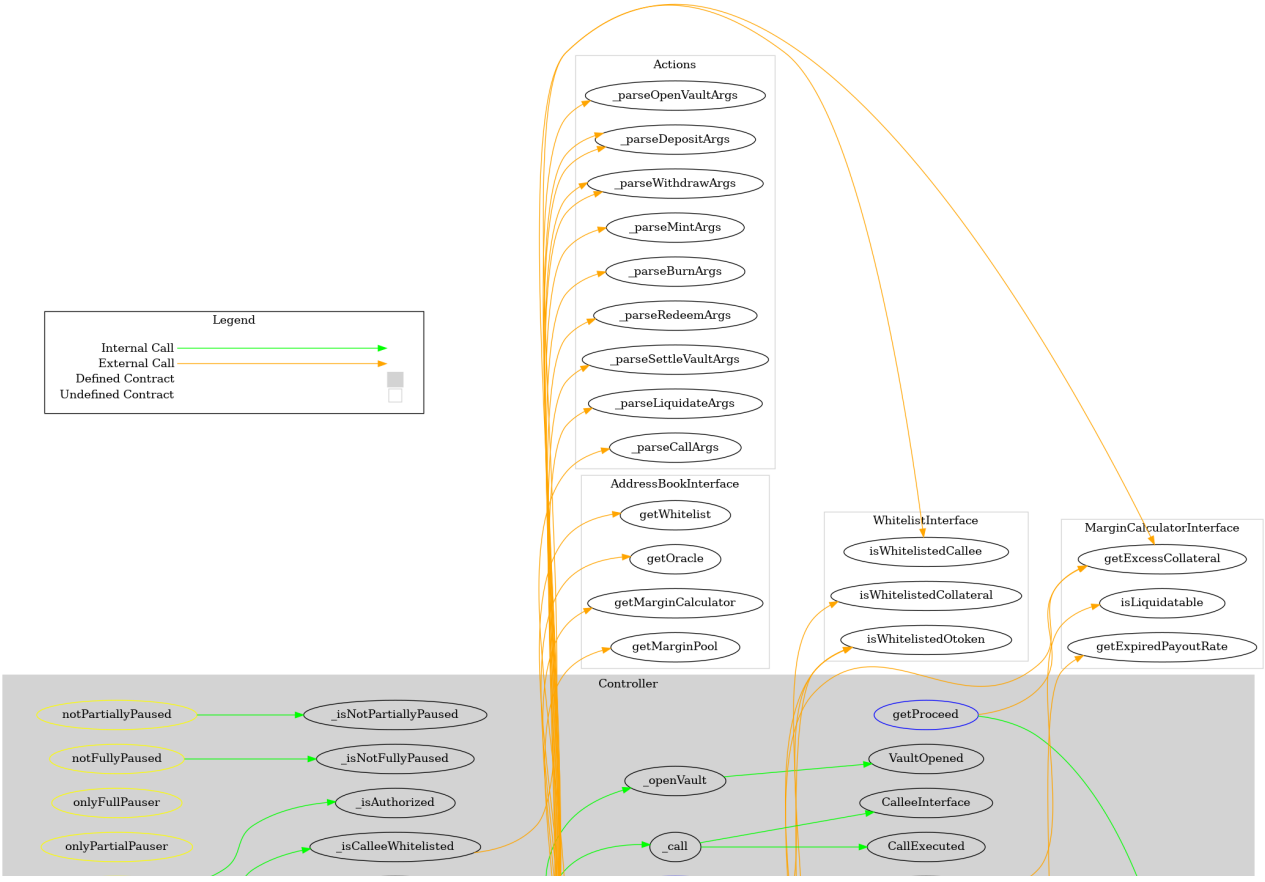
Legend

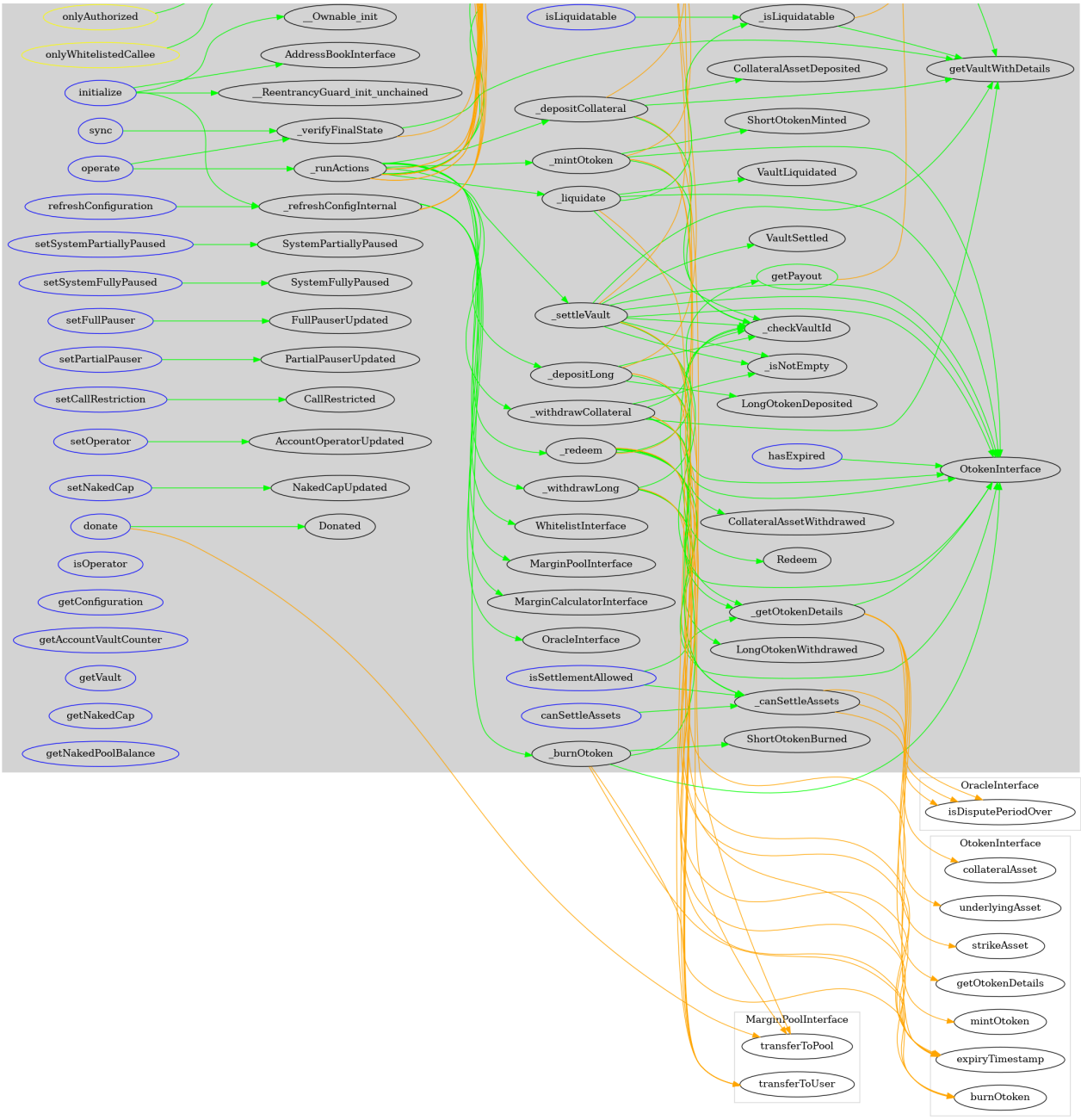
Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Graphs

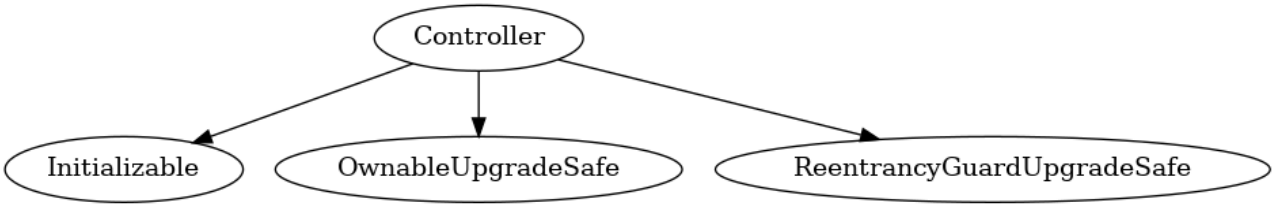
📘 Controller

Execution Graph



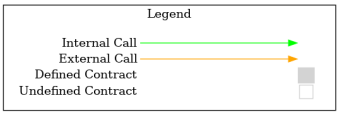


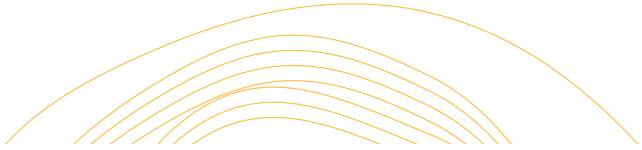
Inheritance

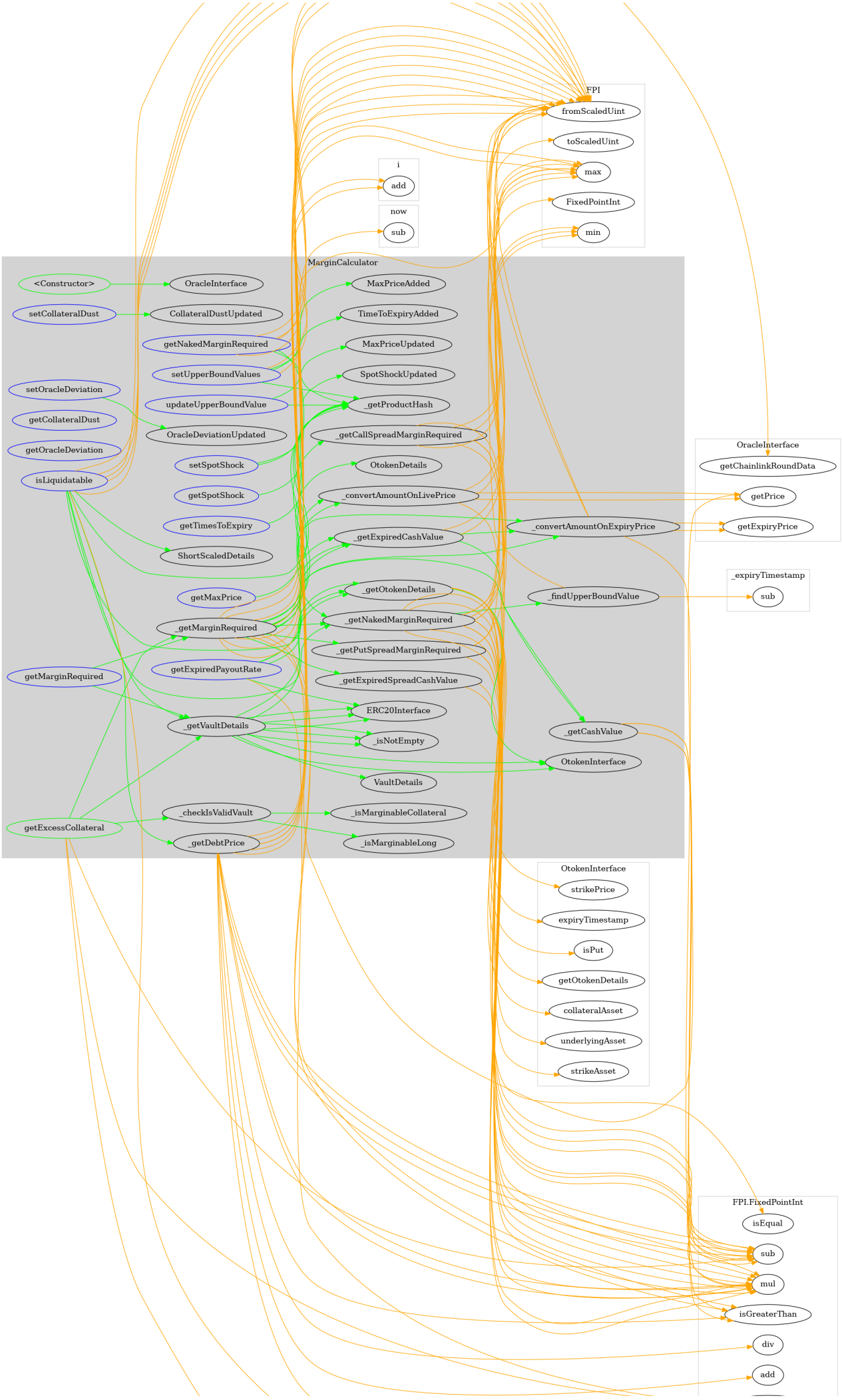


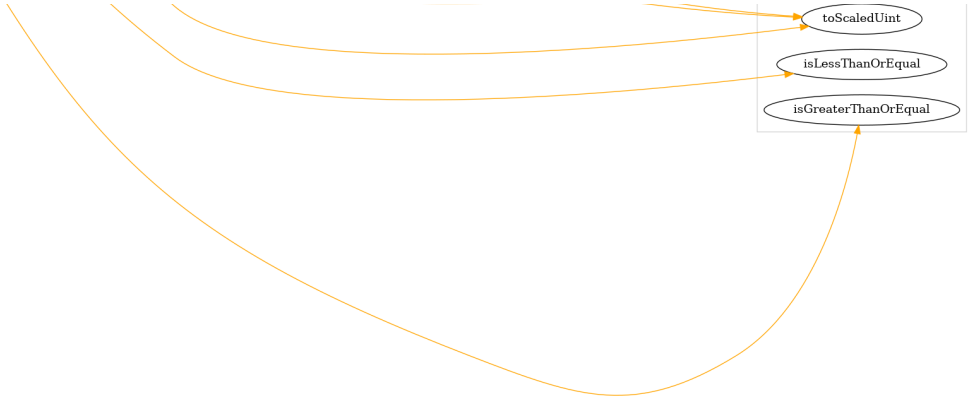
📘 MarginCalculator

Execution Graph

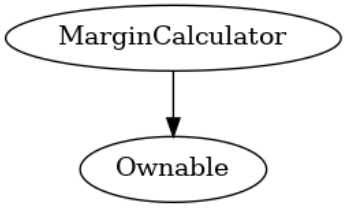





FPI
fromScaledUint
toScaledUint
max
FixedPointInt
min
i
add
now
sub
MarginCalculator
<Constructor>
OracleInterface
setCollateralDust
CollateralDustUpdated
getNakedMarginRequired
setUpperBoundValues
setOracleDeviation
updateUpperBoundValue
getCollateralDust
OracleDeviationUpdated
getOracleDeviation
setSpotShock
isLiquidatable
getSpotShock
getTimesToExpiry
ShortScaledDetails
getMaxPrice
_getMarginRequired
getMarginRequired
getExpiredPayoutRate
_getVaultDetails
getExcessCollateral
_checkIsValidVault
_getDebtPrice
MaxPriceAdded
TimeToExpiryAdded
MaxPriceUpdated
SpotShockUpdated
_getProductHash
_getCallSpreadMarginRequired
OtokenDetails
_convertAmountOnLivePrice
_getExpiredCashValue
_getOtokenDetails
_getNakedMarginRequired
_getPutSpreadMarginRequired
_getExpiredSpreadCashValue
ERC20Interface
_isNotEmpty
VaultDetails
_isMarginableCollateral
_isMarginableLong
OracleInterface
getChainlinkRoundData
getPrice
_convertAmountOnExpiryPrice
getExpiryPrice
_expiryTimestamp
sub
_findUpperBoundValue
_getCashValue
OtokenInterface
OtokenInterface
strikePrice
expiryTimestamp
isPut
getOtokenDetails
collateralAsset
underlyingAsset
strikeAsset
FPI.FixedPointInt
isEqual
sub
mul
isGreaterThan
div
add

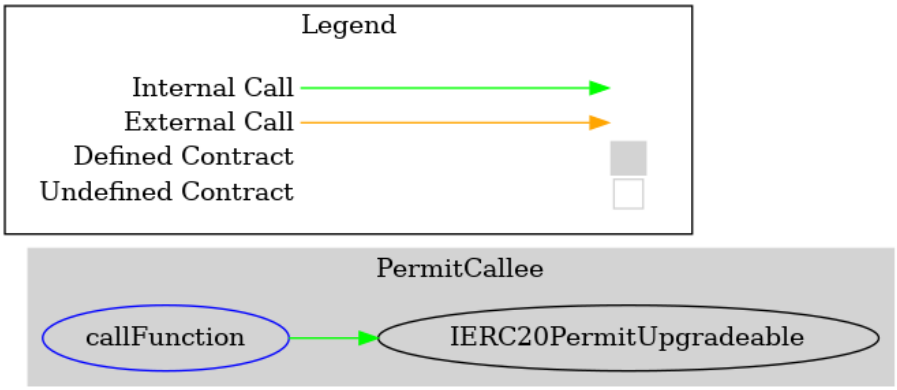


Inheritance

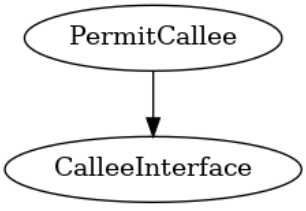


📘 PermitCallee

Execution Graph

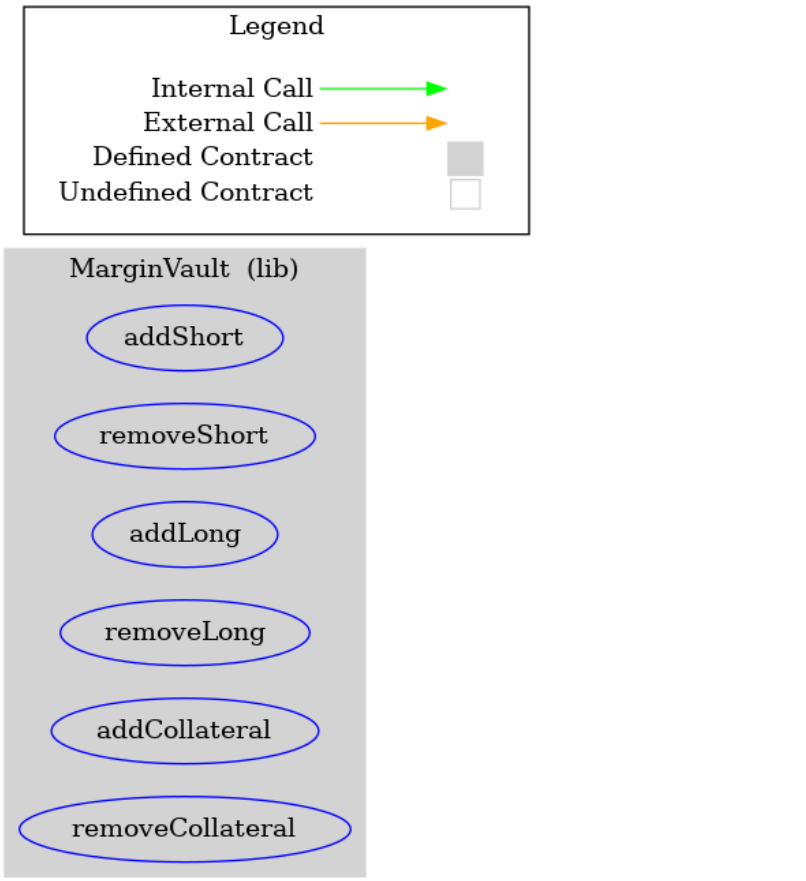


Inheritance



📘 MarginVault

Execution Graph



Inheritance

Describe

📘 Controller

```
$ npx surya describe contracts/core/Controller.sol
 +  Controller (Initializable, OwnableUpgradeSafe, ReentrancyGuardUpgradeSafe)
    - [Int] _isNotPartiallyPaused
    - [Int] _isNotFullyPaused
    - [Int] _isAuthorized
    - [Ext] initialize #
       - modifiers: initializer
    - [Ext] donate #
    - [Ext] setSystemPartiallyPaused #
       - modifiers: onlyPartialPauser
    - [Ext] setSystemFullyPaused #
       - modifiers: onlyFullPauser
    - [Ext] setFullPauser #
       - modifiers: onlyOwner
    - [Ext] setPartialPauser #
       - modifiers: onlyOwner
    - [Ext] setCallRestriction #
       - modifiers: onlyOwner
    - [Ext] setOperator #
```

- [Ext] refreshConfiguration #
  - modifiers: onlyOwner
- [Ext] setNakedCap #
  - modifiers: onlyOwner
- [Ext] operate #
  - modifiers: nonReentrant,notFullyPaused
- [Ext] sync #
  - modifiers: nonReentrant,notFullyPaused
- [Ext] isOperator
- [Ext] getConfiguration
- [Ext] getProceed
- [Ext] isLiquidatable
- [Pub] getPayout
- [Ext] isSettlementAllowed
- [Ext] canSettleAssets
- [Ext] getAccountVaultCounter
- [Ext] hasExpired
- [Ext] getVault
- [Pub] getVaultWithDetails
- [Ext] getNakedCap
- [Ext] getNakedPoolBalance
- [Int] _runActions #
- [Int] _verifyFinalState
- [Int] _openVault #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _depositLong #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _withdrawLong #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _depositCollateral #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _withdrawCollateral #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _mintOtoken #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _burnOtoken #
  - modifiers: notPartiallyPaused,onlyAuthorized
- [Int] _redeem #
- [Int] _settleVault #
  - modifiers: onlyAuthorized
- [Int] _liquidate #
  - modifiers: notPartiallyPaused
- [Int] _call #
  - modifiers: notPartiallyPaused,onlyWhitelistedCallee
- [Int] _checkVaultId
- [Int] _isNotEmpty
- [Int] _isCalleeWhitelisted
- [Int] _isLiquidatable
- [Int] _getOtokenDetails
- [Int] _canSettleAssets
- [Int] _refreshConfigInternal #

```
($) = payable function
# = non-constant function
```

📘 MarginCalculator

```
$ npx surya describe ./core/MarginCalculator.sol
 +  MarginCalculator (Ownable)
    - [Pub] <Constructor> #
    - [Ext] setCollateralDust #
       - modifiers: onlyOwner
    - [Ext] setUpperBoundValues #
       - modifiers: onlyOwner
    - [Ext] updateUpperBoundValue #
       - modifiers: onlyOwner
    - [Ext] setSpotShock #
       - modifiers: onlyOwner
    - [Ext] setOracleDeviation #
       - modifiers: onlyOwner
    - [Ext] getCollateralDust
    - [Ext] getTimesToExpiry
    - [Ext] getMaxPrice
    - [Ext] getSpotShock
    - [Ext] getOracleDeviation
    - [Ext] getNakedMarginRequired
    - [Ext] getExpiredPayoutRate
    - [Ext] isLiquidatable
    - [Ext] getMarginRequired
    - [Pub] getExcessCollateral
    - [Int] _getExpiredCashValue
    - [Int] _getMarginRequired
    - [Int] _getNakedMarginRequired
    - [Int] _findUpperBoundValue
    - [Int] _getPutSpreadMarginRequired
    - [Int] _getCallSpreadMarginRequired
    - [Int] _convertAmountOnLivePrice
    - [Int] _convertAmountOnExpiryPrice
    - [Int] _getDebtPrice
    - [Int] _getVaultDetails
    - [Int] _getExpiredSpreadCashValue
    - [Int] _isNotEmpty
    - [Int] _checkIsValidVault
    - [Int] _isMarginableLong
    - [Int] _isMarginableCollateral
    - [Int] _getProductHash
    - [Int] _getCashValue
    - [Int] _getOtokenDetails


($) = payable function
# = non-constant function
```

📘 PermitCallee

```
$ npx surya describe ./external/callees/PermitCallee.sol
 +  PermitCallee (CalleeInterface)
    - [Ext] callFunction #


 ($) = payable function
 # = non-constant function
```

📘 MarginVault

```
$ npx surya describe ./libs/MarginVault.sol
 + [Lib] MarginVault
    - [Ext] addShort #
    - [Ext] removeShort #
    - [Ext] addLong #
    - [Ext] removeLong #
    - [Ext] addCollateral #
    - [Ext] removeCollateral #


 ($) = payable function
 # = non-constant function
```

License