AKIRA TECH

PROJECT

Volt Protocol Core

CLIENT

Volt Protocol

DATE

September 2022

REVIEWERS

Daniel Luca

@cleanunicorn

Table of Contents

Details

- **Client** Volt Protocol
- **Date** September 2022
- **Lead reviewer** Daniel Luca (@cleanunicorn)
- **Reviewers** Daniel Luca (@cleanunicorn)
- **Repository:** Volt Protocol Core
- **Commit hash** `86868c6f77fca67ed6586674437cc9bfc85c1963`
- **Final commit hash** `1a1ce42f5131059084b8def03a75c334762fad98`
- **Technologies**
    - Solidity
    - Node.JS

Issues Summary

SEVERITY	OPEN	CLOSED
Informational	0	1
Minor	3	0
Medium	2	0
Major	0	0

Executive summary

This report represents the results of the engagement with **Volt Protocol** to review **Volt Protocol Core**.

The review was conducted over the course of **1 week** from **September 12 to September 16, 2022**. A total of **5 person-days** were spent reviewing the code.

Week 1

During the first week, we started to manually review the code and the provided documents. The code is well documented and of good quality, but it's highly modular and it's hard to understand the overall architecture. We started to write a high-level architecture diagram and we started to write a list of the most important contracts.

We identified a few low severity issues, but we didn't find any critical issues. We also identified a few issues that could be improved, but they don't represent a security risk.

On Wednesday, we set up a meeting with the client to discuss the architecture and the issues we found. The dev team needed us to check a new PR that is about to hit the mainnet. This PR is identified by commit hash `1a1ce42f5131059084b8def03a75c334762fad98`. The scope was extended to include the file `contracts/pcv/utils/ERC20Allocator.sol`.

On Thursday, we focused on the newly added file. We had another meeting with the client where we discussed details in the implementation of the new contract.

On Friday, we made sure we didn't have any open or unfinished issues and had another meeting with the client to deliver the report and discuss any open questions.

Scope

The initial review focused on the Volt Protocol Core repository, identified by the commit hash `86868c6f77fca67ed6586674437cc9bfc85c1963`. Later during the week, another file was added to the scope, identified by the commit hash `1a1ce42f5131059084b8def03a75c334762fad98`.

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:

- PriceBoundPSM
- Core
- L2Core
- VoltSystemOracle
- OraclePassThrough
- TimelockController - from OpenZeppelin
- PCVGuardian
- PCVGuardAdmin
- CoreRef
- OracleRef
- ERC20CompoundPCVDeposit
- ERC20Allocator - added later in the scope

Documents:

- Market Governance Whitepaper
- Market Governance Alpha

Issues

Oracle implementation can be made more robust

Status Acknowledged Severity Informational

Description

The current oracle implementation does not account for external systems failures. There is no immediate threat, but the oracle system must be a very robust and trusted component.

Consider this approach for implementing an oracle system.

We define a base class that handles passing the value obtained from an oracle implementation. This class implements a robust approach for getting the value, even if the implementation fails.

```
abstract contract OracleBase {
    function value() external returns (uint256 value_, bool valid_) {
        // Try to get the value
        try this.obtainValue() returns (uint256 valueReturned, bool valid) {
            // Prepare the value to be returned
            value_ = valueReturned;
            // Pass the valid bool
            valid_ = valid;
        } catch {
            value_ = 0;
            valid_ = false;
        }

        return (value_, valid_);
    }

    function obtainValue() external virtual returns (uint256 value, bool valid);
}
```

The call to `obtainValue()` can fail because of external calls or improper developer implementation. Whatever the case, the base class can handle these fails and can do much more if multiple fails happen in a row, or provide a cached value, or notify a system that they're offline.

Any oracle will use this base class and must implement `obtainValue` specific to the value they need to retrieve.

```

contract OracleImplementation is OracleBase {
    ExternalContract immutable public ec;

    constructor(ExternalContract _ec) {
        // Save the external provider's address
        ec = _ec;
    }

    function obtainValue() override(OracleBase) external returns (uint256, bool) {
        // Make sure this is not called externally by anyone else.
        require(msg.sender == address(this), "Oracle must call itself to get the value");

        // Talk to `ExternalContract` and get the scale
        uint256 scale = ec.getScale();

        // Sometimes you need to do something with the value
        scale = scale / 10;

        // Validate the processed value
        bool valid = scale <= 42;
        // Return scale
        return (scale, valid);
    }
}
```

In this case, `OracleImplementation` talks to an external system, and they don't need to account for any failures, unexpected upgrades or weird behavior.

Its only purpose is to obtain the value, process it (if necessary), validate it and return the value if the validity.

`OracleBase` should not care what the value represents, its purpose is to handle failures and pass on the values.

`OracleImplementation` has to understand the value obtained and must know how to process and validate it.

Finally, we tested this with an implementation that returns a value or sometimes fails to ensure our example implementation holds.

```
contract ExternalContract {
    function getScale() public view returns (uint256 scale_) {
        // Simulate random reverts
        if (block.number % 2 == 0) {
            // fail
            revert("Sometimes it fails");
        } else {
            // return a value, do not fail
            return 12;
        }
    }
}
```

Separating the concerns between `OracleBase` and `OracleImplementation` allows for:

- adding new developers that do not have to understand the whole system in order to develop an oracle;
- protecting from unexpected external changes or upgrades;
- protecting from unexpected external fails.

Recommendation

Consider stabilizing the oracle interface and implementation to allow for an increase in developer speed, while retaining robust principles and execution

Setting the backup oracle should impose the same restrictions everywhere

Status Open Severity Medium

Description

The `OracleRef` contract is deployed with a few arguments:

code/contracts/refs/OracleRef.sol#L27-L39

```
    /// @notice OracleRef constructor
    /// @param _core Fei Core to reference
    /// @param _oracle oracle to reference
    /// @param _backupOracle backup oracle to reference
    /// @param _decimalsNormalizer number of decimals to normalize the oracle feed if necessary
    /// @param _doInvert invert the oracle price if this flag is on
    constructor(
        address _core,
        address _oracle,
        address _backupOracle,
        int256 _decimalsNormalizer,
        bool _doInvert
    ) CoreRef(_core) {
```

Before setting the backup oracle, its value is checked to be different from `address(0)` and different from `_oracle`:

code/contracts/refs/OracleRef.sol#L41-L43

```
        if (_backupOracle != address(0) && _backupOracle != _oracle) {
            _setBackupOracle(_backupOracle);
        }
```

However, when calling the external method `setBackupOracle` the same checks are not applied:

code/contracts/refs/OracleRef.sol#L70-L78

```
    /// @notice sets the referenced backup oracle
    /// @param newBackupOracle the new backup oracle to reference
    function setBackupOracle(address newBackupOracle)
        external
        override
        onlyGovernorOrAdmin
    {
        _setBackupOracle(newBackupOracle);
    }
```

code/contracts/refs/OracleRef.sol#L133-L138

```
    // Supports zero address if no backup
    function _setBackupOracle(address newBackupOracle) internal {
        address oldBackupOracle = address(backupOracle);
        backupOracle = IOracle(newBackupOracle);
        emit BackupOracleUpdate(oldBackupOracle, newBackupOracle);
    }
```

We can see the comment saying that the address zero can be set if we want to disable the oracle:

code/contracts/refs/OracleRef.sol#L133

```
    // Supports zero address if no backup
```

Which disables the backup oracle in `readOracle`:

code/contracts/refs/OracleRef.sol#L103

```
        if (!valid && address(backupOracle) != address(0)) {
```

The second check is not enforced, thus the backup oracle could be equal to the main oracle.

Recommendation

Add a check to ensure that the backup oracle is different from the main oracle in the internal method `_setBackupOracle` and remove the same check in `constructor`.

Setting the invert flag in `OracleRef` might have unexpected consequences

Status Open Severity Medium

Description

The method `setDoInvert` sets a storage flag that affects the value read from the oracle.

The value determines if it is inverted according to `doInvert`

$$f(x) = \begin{cases} 1/x & \text{when doInvert is } \textit{true} \\ x & \text{otherwise} \end{cases}$$

code/contracts/refs/OracleRef.sol#L108-L111

```
        // Invert the oracle price if necessary
        if (doInvert) {
            _peg = invert(_peg);
        }
```

The value then proceeds to be scaled before being returned:

code/contracts/refs/OracleRef.sol#L113-L123

```
        // Scale the oracle price by token decimals delta if necessary
        uint256 scalingFactor;
        if (decimalsNormalizer < 0) {
            scalingFactor = 10**(-1 * decimalsNormalizer).toUint256();
            _peg = _peg.div(scalingFactor);
        } else {
            scalingFactor = 10**decimalsNormalizer.toUint256();
            _peg = _peg.mul(scalingFactor);
        }

        return _peg;
```

The storage slot `doInvert` is updated by calling the `external` method:

code/contracts/refs/OracleRef.sol#L54-L58

```
    /// @notice sets the flag for whether to invert or not
    /// @param newDoInvert the new flag for whether to invert
    function setDoInvert(bool newDoInvert) external override onlyGovernor {
        _setDoInvert(newDoInvert);
    }
```

This proceeds to call internally `_setDoInvert`:

code/contracts/refs/OracleRef.sol#L140-L149

```
    function _setDoInvert(bool newDoInvert) internal {
        bool oldDoInvert = doInvert;
        doInvert = newDoInvert;

        if (oldDoInvert != newDoInvert) {
            _setDecimalsNormalizer(-1 * decimalsNormalizer);
        }

        emit InvertUpdate(oldDoInvert, newDoInvert);
    }
```

It is not obvious nor documented that changing the invert value will also update the decimals normalizer.

Consider the example where the governance wants to update the oracle's invert and decimal scaling. To do that the following methods need to be executed:

- `OracleRef.setDoInvert(bool)`
- `OracleRef.setDecimalsNormalizer(int256)`

Depending on the order of the called methods you will have different final values for the decimals normalizer.

1. Updating `doInvert` first and `decimalsNormalizer` second

Initial values

```
doInvert = false;
decimalsNormalizer = 18;
```

Update values

```
doInvert(true);
setDecimalsNormalizer(-12);
```

Final values

```
doInvert = true;
decimalsNormalizer = -12;
```

2. Updating `decimalsNormalizer` first and `doInvert` second

We start from the same initial values

```
doInvert = false;
decimalsNormalizer = 18;
```

We update the values, but in a different order

```
setDecimalsNormalizer(-12);
doInvert(true);
```

The final values can be unexpected for the governance

```
doInvert = true;
decimalsNormalizer = 12;
```

Even though we set the decimals to be `-12`, because we also updated the invert flag, we obtained a different value for our decimals scaling. The order of operations determines our final value for the decimals scaling.

This time the `decimalsNormalizer` is updated first. Thus, we have `decimalsNormalizer = -12`.

When the method `doInvert(true)` is executed, the old value is first saved, and the flag `doInvert` is set to `true`:

code/contracts/refs/OracleRef.sol#L140-L142

```
    function _setDoInvert(bool newDoInvert) internal {
        bool oldDoInvert = doInvert;
        doInvert = newDoInvert;
```

After this, because the old value is different from the current value, the decimals normalizer is also inverted:

code/contracts/refs/OracleRef.sol#L144-L146

```
        if (oldDoInvert != newDoInvert) {
            _setDecimalsNormalizer(-1 * decimalsNormalizer);
        }
```

This might be unexpected because it's not obvious, it's "hidden" in the internal method and an event is not emitted.

We can clearly see how changing the order of the operations can unexpectedly update `decimalsNormalizer`.

Recommendation

Consider removing the logic related to updating `decimalsNormalizer` in the internal method `_setDoInvert` :

code/contracts/refs/OracleRef.sol#L144-L146

```
        if (oldDoInvert != newDoInvert) {
            _setDecimalsNormalizer(-1 * decimalsNormalizer);
        }
```

Method `compoundInterest` can be gas optimized when emitting event

Status Open Severity Minor

Description

The method `compountInterest` calculates the compounded interest if enough time has passed.

If the conditions are met, a new oracle price is saved and an event is emitted.

code/contracts/oracle/VoltSystemOracle.sol#L80-L88

```
        /// first set Oracle Price to interpolated value
        oraclePrice = getCurrentOraclePrice();

        /// set periodStartTime to periodStartTime + timeframe,
        /// this is equivalent to init timed, which wipes out all unaccumulated compounded inter
        /// and cleanly sets the start time.
        periodStartTime = periodEndTime;

        emit InterestCompounded(periodStartTime, oraclePrice);
```

The emission of the event can be gas optimized a bit, by using the stack value instead of the storage slot since they represent the same value.

code/contracts/oracle/VoltSystemOracle.sol#L83-L88

```
        /// set periodStartTime to periodStartTime + timeframe,
        /// this is equivalent to init timed, which wipes out all unaccumulated compounded inter
        /// and cleanly sets the start time.
        periodStartTime = periodEndTime;

        emit InterestCompounded(periodStartTime, oraclePrice);
```

Recommendation

Consider emitting the stack value `periodEndTime` instead using the storage slot `periodStartTime` since they have the same value and will use less gas.

`_validPrice` can include floor and ceiling values when checking the validity

Status Open Severity Minor

Description

A price is considered valid if the internal method `_validPrice` returns true.

code/contracts/peg/PriceBoundPSM.sol#L122-L135

```
    /// @notice helper function to determine if price is within a valid range
    function _validPrice(Decimal.D256 memory price)
        internal
        view
        returns (bool valid)
    {
        valid =
            price.greaterThan(
                Decimal.ratio(floor, Constants.BASIS_POINTS_GRANULARITY)
            ) &&
            price.lessThan(
                Decimal.ratio(ceiling, Constants.BASIS_POINTS_GRANULARITY)
            );
    }
```

This method checks if the provided price is strictly greater than the floor and strictly less than the floor.

code/contracts/peg/PriceBoundPSM.sol#L129-L134

```
            price.greaterThan(
                Decimal.ratio(floor, Constants.BASIS_POINTS_GRANULARITY)
            ) &&
            price.lessThan(
                Decimal.ratio(ceiling, Constants.BASIS_POINTS_GRANULARITY)
            );
```

These methods are taken from the `Decimal` library.

code/contracts/external/Decimal.sol#L163-L177

```
    function greaterThan(D256 memory self, D256 memory b)
        internal
        pure
        returns (bool)
    {
        return compareTo(self, b) == 2;
    }

    function lessThan(D256 memory self, D256 memory b)
        internal
        pure
        returns (bool)
    {
        return compareTo(self, b) == 0;
    }
```

Using a strictly greater/less check will exclude the values of floor and ceiling as being valid. This forces the governance to add floor and ceiling values similar to `floor = 0.7999999999999` and `ceiling = 1.199999999999` to describe a valid interval from `0.8` to `1.2`, included.

If the check would accept the limits as valid, the governance could set `floor = 0.8` and `ceiling = 1.2`, which are a lot easier to reason about and verify.

Recommendation

Consider using the methods `greaterThanOrEqualTo` and `lessThanOrEqualTo` provided by `Decimal` to avoid using awkward values.

code/contracts/external/Decimal.sol#L179-L193

```
function greaterThanOrEqualTo(D256 memory self, D256 memory b)
    internal
    pure
    returns (bool)
{
    return compareTo(self, b) > 0;
}

function lessThanOrEqualTo(D256 memory self, D256 memory b)
    internal
    pure
    returns (bool)
{
    return compareTo(self, b) < 2;
}
```

[optional] References

Setting ceiling basis points does not need a non zero validation

Status Open Severity Minor

Description

A governor or an admin can set the price floor and the ceiling between which swaps are enabled.

To set the ceiling, they call `setOracleCeilingBasisPoints`

code/contracts/peg/PriceBoundPSM.sol#L62-L69

```
/// @notice sets the ceiling price in BP
function setOracleCeilingBasisPoints(uint256 newCeilingBasisPoints)
    external
    override
    onlyGovernorOrAdmin
{
    _setCeilingBasisPoints(newCeilingBasisPoints);
}
```

The internal method `_setCeilingBasisPoints` make sure the provided value is

- non zero

code/contracts/peg/PriceBoundPSM.sol#L84-L87

```
        require(
            newCeilingBasisPoints != 0,
            "PegStabilityModule: invalid ceiling"
        );
```

- greater than the floor

code/contracts/peg/PriceBoundPSM.sol#L88-L98

```
        require(
            Decimal
                .ratio(
                    newCeilingBasisPoints,
                    Constants.BASIS_POINTS_GRANULARITY
                )
                .greaterThan(
                    Decimal.ratio(floor, Constants.BASIS_POINTS_GRANULARITY)
                ),
            "PegStabilityModule: ceiling must be greater than floor"
        );
```

The check that makes sure the value is non-zero is not required since the ceiling has to be greater than the floor, and the floor has to be greater than zero:

code/contracts/peg/PriceBoundPSM.sol#L105-L107

```
    /// @notice helper function to set the floor in basis points
    function _setFloorBasisPoints(uint256 newFloorBasisPoints) internal {
        require(newFloorBasisPoints != 0, "PegStabilityModule: invalid floor");
```

To increase readability and to help the developer reason about the code, the check can stay there. Still, to slightly decrease the gas cost, a comment can be added that explains why the non-zero check is not needed, retaining readability.

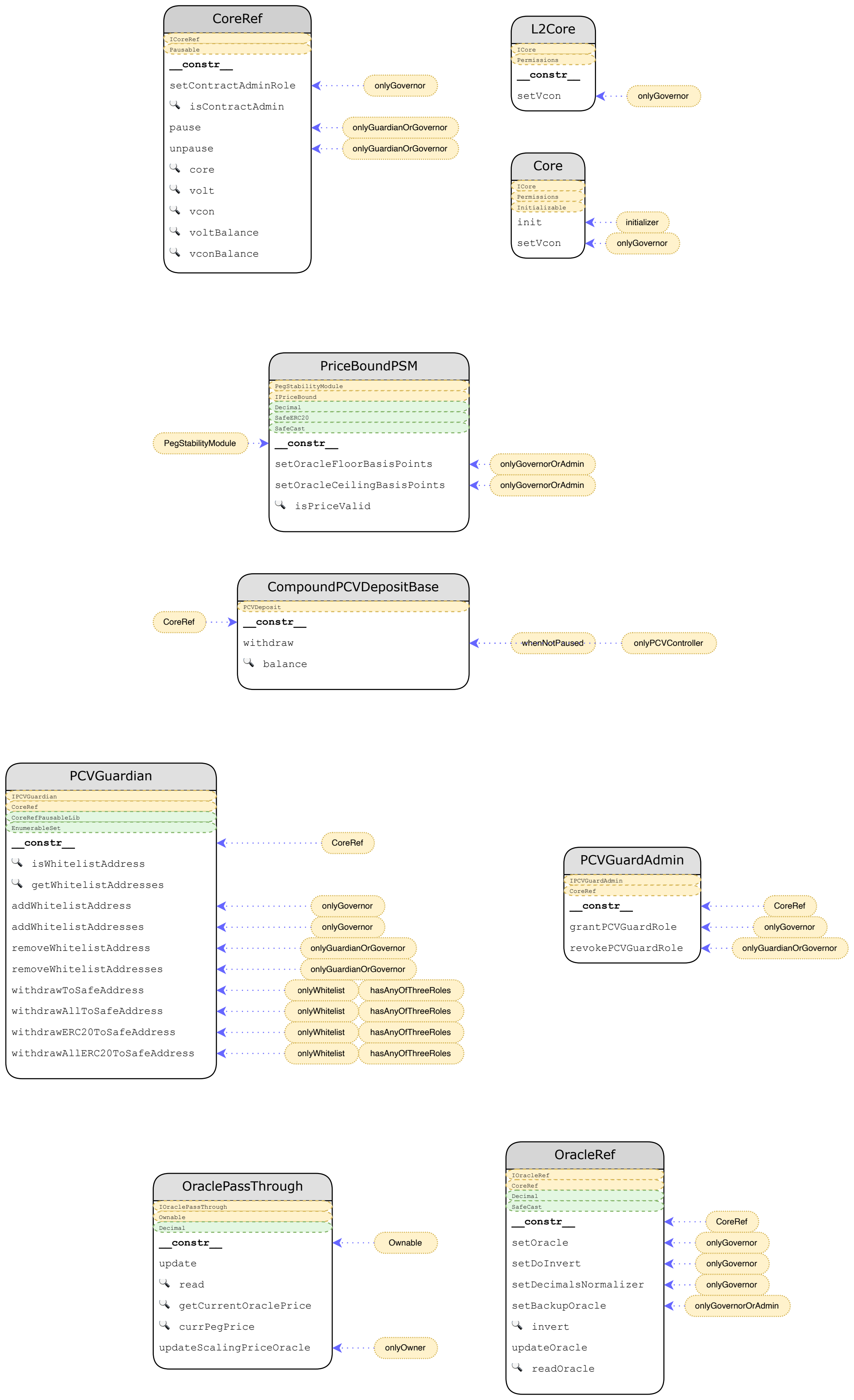
Recommendation

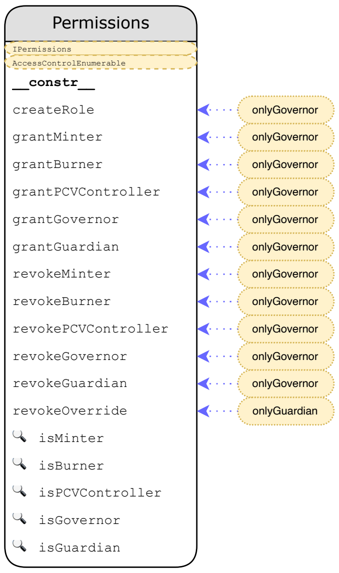
Consider replacing the `require` with a comment that explains why the check is not necessary.

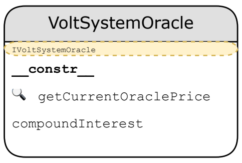
Artifacts

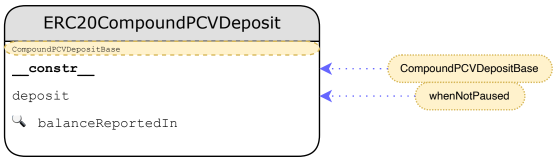
Architecture

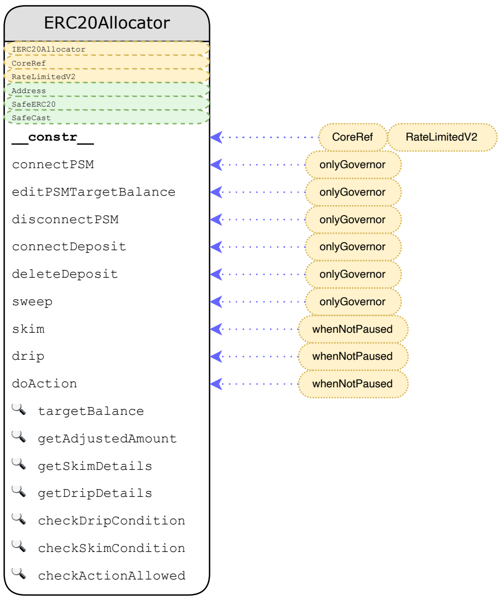
The architecture is extremely hard to understand and it's not well documented. We started to work on a high-level architecture diagram.











Describe

We used surya to generate these outputs:

```
$ npx surya describe ./Contract.sol
```

Core

```
 +  Core (ICore, Permissions, Initializable)
    - [Ext] init #
       - modifiers: initializer
    – [Ext] setVcon #
       – modifiers: onlyGovernor
```

PriceBoundPSM

```
 +  PriceBoundPSM (PegStabilityModule, IPriceBound)
    – [Pub] <Constructor> #
       – modifiers: PegStabilityModule
    – [Ext] setOracleFloorBasisPoints #
       – modifiers: onlyGovernorOrAdmin
    – [Ext] setOracleCeilingBasisPoints #
       – modifiers: onlyGovernorOrAdmin
    – [Ext] isPriceValid
    – [Int] _allocate #
    – [Int] _setCeilingBasisPoints #
    – [Int] _setFloorBasisPoints #
    – [Int] _validPrice
    – [Int] _validatePriceRange
```

L2Core

```
 +  L2Core (ICore, Permissions)
    – [Pub] <Constructor> #
    – [Ext] setVcon #
       – modifiers: onlyGovernor
```

VoltSystemOracle

```
 +  VoltSystemOracle (IVoltSystemOracle)
    – [Pub] <Constructor> #
    – [Pub] getCurrentOraclePrice
    – [Ext] compoundInterest #
```

OraclePassThrough

```
 +  OraclePassThrough (IOraclePassThrough, Ownable)
    – [Pub] <Constructor> #
       – modifiers: Ownable
    – [Pub] update #
    – [Ext] read
    – [Ext] getCurrentOraclePrice
    – [Ext] currPegPrice
    – [Ext] updateScalingPriceOracle #
       – modifiers: onlyOwner
```

PCVGuardian

```
+  PCVGuardian (IPCVGuardian, CoreRef)
   - [Pub] <Constructor> #
      - modifiers: CoreRef
   - [Pub] isWhitelistAddress
   - [Pub] getWhitelistAddresses
   - [Ext] addWhitelistAddress #
      - modifiers: onlyGovernor
   - [Ext] addWhitelistAddresses #
      - modifiers: onlyGovernor
   - [Ext] removeWhitelistAddress #
      - modifiers: onlyGuardianOrGovernor
   - [Ext] removeWhitelistAddresses #
      - modifiers: onlyGuardianOrGovernor
   - [Ext] withdrawToSafeAddress #
      - modifiers: hasAnyOfThreeRoles,onlyWhitelist
   - [Ext] withdrawAllToSafeAddress #
      - modifiers: hasAnyOfThreeRoles,onlyWhitelist
   - [Ext] withdrawERC20ToSafeAddress #
      - modifiers: hasAnyOfThreeRoles,onlyWhitelist
   - [Ext] withdrawAllERC20ToSafeAddress #
      - modifiers: hasAnyOfThreeRoles,onlyWhitelist
   - [Int] _withdrawToSafeAddress #
   - [Int] _withdrawERC20ToSafeAddress #
   - [Int] _addWhitelistAddress #
   - [Int] _removeWhitelistAddress #
```

PCVGuardAdmin

```
+  PCVGuardAdmin (IPCVGuardAdmin, CoreRef)
   - [Pub] <Constructor> #
      - modifiers: CoreRef
   - [Ext] grantPCVGuardRole #
      - modifiers: onlyGovernor
   - [Ext] revokePCVGuardRole #
      - modifiers: onlyGuardianOrGovernor
```

CoreRef

```
+  CoreRef (ICoreRef, Pausable)
   - [Pub] <Constructor> #
   - [Int] _initialize #
   - [Ext] setContractAdminRole #
      - modifiers: onlyGovernor
   - [Pub] isContractAdmin
   - [Pub] pause #
      - modifiers: onlyGuardianOrGovernor
   - [Pub] unpause #
      - modifiers: onlyGuardianOrGovernor
   - [Pub] core
   - [Pub] volt
```

```
  - [Pub] vcon
  - [Pub] voltBalance
  - [Pub] vconBalance
  - [Int] _burnVoltHeld #
  - [Int] _mintVolt #
  - [Int] _setContractAdminRole #
```

OracleRef

```
+  OracleRef (IOracleRef, CoreRef)
   - [Pub] <Constructor> #
      - modifiers: CoreRef
   - [Ext] setOracle #
      - modifiers: onlyGovernor
   - [Ext] setDoInvert #
      - modifiers: onlyGovernor
   - [Ext] setDecimalsNormalizer #
      - modifiers: onlyGovernor
   - [Ext] setBackupOracle #
      - modifiers: onlyGovernorOrAdmin
   - [Pub] invert
   - [Pub] updateOracle #
   - [Pub] readOracle
   - [Int] _setOracle #
   - [Int] _setBackupOracle #
   - [Int] _setDoInvert #
   - [Int] _setDecimalsNormalizer #
   - [Int] _setDecimalsNormalizerFromToken #
```

ERC20CompoundPCVDeposit

```
+  ERC20CompoundPCVDeposit (CompoundPCVDepositBase)
   - [Pub] <Constructor> #
      - modifiers: CompoundPCVDepositBase
   - [Ext] deposit #
      - modifiers: whenNotPaused
   - [Int] _transferUnderlying #
   - [Pub] balanceReportedIn
```

Legend

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

Tests

```
$ npm run test
```

```
> @voltprotocol/volt-protocol-core@1.0.0 test
> forge test --match-contract UnitTest -vvv

[⠒] Compiling...
[⠑] Compiling 1 files with 0.4.26
[⠘] Compiling 71 files with 0.8.13
[⠃] Compiling 78 files with 0.8.16
[⠔] Solc 0.4.26 finished in 36.38ms
[⠊] Solc 0.8.16 finished in 4.48s
[⠘] Solc 0.8.13 finished in 10.90s
Compiler run successful (with warnings)
contracts/external/WETH9.sol:47:9: Warning: Invoking events without "emit" prefix is deprecated.
        Deposit(msg.sender, msg.value);
        ^----------------------------^



contracts/external/WETH9.sol:54:9: Warning: Invoking events without "emit" prefix is deprecated.
        Withdrawal(msg.sender, wad);
        ^-------------------------^



contracts/external/WETH9.sol:58:16: Warning: Using contract member "balance" inherited from the
        return this.balance;
               ^----------^



contracts/external/WETH9.sol:63:9: Warning: Invoking events without "emit" prefix is deprecated.
        Approval(msg.sender, guy, wad);
        ^----------------------------^



contracts/external/WETH9.sol:86:9: Warning: Invoking events without "emit" prefix is deprecated.
        Transfer(src, dst, wad);
        ^---------------------^



warning[8760]: Warning: This declaration has the same name as another declaration.
  --> contracts/pcv/utils/ERC20Allocator.sol:79:9:
   |
79 |         uint248 targetBalance,
   |         ^^^^^^^^^^^^^^^^^^^^^
Note: The other declaration is here:
   --> contracts/pcv/utils/ERC20Allocator.sol:280:5:
    |
280 |     function targetBalance(address psm) external view returns (uint256) {
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[8760]: Warning: This declaration has the same name as another declaration.
```

```
   --> contracts/pcv/utils/ERC20Allocator.sol:104:48:
    |
104 |     function editPSMTargetBalance(address psm, uint248 targetBalance)
    |                                                ^^^^^^^^^^^^^^^^^^^^^
Note: The other declaration is here:
   --> contracts/pcv/utils/ERC20Allocator.sol:280:5:
    |
280 |     function targetBalance(address psm) external view returns (uint256) {
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[2072]: Warning: Unused local variable.
  --> contracts/test/integration/vip/vip8.sol:94:9:
   |
94 |         uint256 daiBalance = PegStabilityModule(MainnetAddresses.FEI_DAI_PSM)
   |         ^^^^^^^^^^^^^^^^^^



warning[2072]: Warning: Unused local variable.
   --> contracts/test/unit/pcv/utils/ERC20Allocator.t.sol:619:9:
    |
619 |         uint256 skimAmount = (bufferCap - bufferEnd) * 2;
    |         ^^^^^^^^^^^^^^^^^^



warning[2018]: Warning: Function state mutability can be restricted to view
   --> contracts/test/integration/vip/vip10.sol:121:5:
    |
121 |     function getMainnetProposal()
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[2018]: Warning: Function state mutability can be restricted to pure
   --> contracts/test/integration/vip/vip10.sol:185:5:
    |
185 |     function arbitrumSetup() public override {
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[2018]: Warning: Function state mutability can be restricted to pure
   --> contracts/test/integration/vip/vip10.sol:189:5:
    |
189 |     function arbitrumValidate() public override {
    |     ^ (Relevant source part starts here and spans across multiple lines).
```

```
warning[2018]: Warning: Function state mutability can be restricted to pure
   --> contracts/test/integration/vip/vip8.sol:141:5:
    |
141 |     function arbitrumSetup() public override {
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[2018]: Warning: Function state mutability can be restricted to pure
   --> contracts/test/integration/vip/vip8.sol:145:5:
    |
145 |     function arbitrumValidate() public override {
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[2018]: Warning: Function state mutability can be restricted to pure
   --> contracts/test/integration/vip/vip9.sol:110:5:
    |
110 |     function arbitrumSetup() public override {
    |     ^ (Relevant source part starts here and spans across multiple lines).



warning[2018]: Warning: Function state mutability can be restricted to pure
   --> contracts/test/integration/vip/vip9.sol:114:5:
    |
114 |     function arbitrumValidate() public override {
    |     ^ (Relevant source part starts here and spans across multiple lines).




Running 2 tests for contracts/test/unit/Volt.t.sol:UnitTestVolt
[PASS] testDeployedMetaData() (gas: 17541)
[PASS] testMintsVolt() (gas: 70031)
Test result: ok. 2 passed; 0 failed; finished in 8.31ms

Running 2 tests for contracts/test/unit/core/Core.t.sol:UnitTestCore
[PASS] testGovernorSetsVcon() (gas: 22417)
[PASS] testNonGovernorFailsSettingVcon() (gas: 13096)
Test result: ok. 2 passed; 0 failed; finished in 7.28ms

Running 4 tests for contracts/test/unit/oracle/OraclePassThrough.t.sol:UnitTestOraclePassThrough
[PASS] testDataPassThroughSync() (gas: 23897)
[PASS] testSetup() (gas: 12820)
[PASS] testUpdateScalingPriceOracleFailureNotGovernor() (gas: 13697)
[PASS] testUpdateScalingPriceOracleSuccess() (gas: 244964)
Test result: ok. 4 passed; 0 failed; finished in 10.97ms
```

```
Running 6 tests for contracts/test/unit/utils/KArrayTree.t.sol:KArrayTreeUnitTest
[PASS] testAddDuplicateFails() (gas: 5643)
[PASS] testAddDuplicateFailsFind() (gas: 39737)
[PASS] testCanChangeRole() (gas: 40888)
[PASS] testCannotChangeToExistingRole() (gas: 5672)
[PASS] testFree() (gas: 52596)
[PASS] testSetup() (gas: 46808)
Test result: ok. 6 passed; 0 failed; finished in 452.71µs

Running 3 tests for contracts/test/unit/utils/Deviation.t.sol:UnitTestDeviation
[PASS] testDeviation() (gas: 1332)
[PASS] testOutsideDeviation() (gas: 3350)
[PASS] testWithinDeviation() (gas: 3120)
Test result: ok. 3 passed; 0 failed; finished in 179.65µs

Running 3 tests for contracts/test/unit/core/L2Core.t.sol:UnitTestL2Core
[PASS] testGovernorSetsVcon() (gas: 39600)
[PASS] testNonGovernorFailsSettingVcon() (gas: 13129)
[PASS] testSetup() (gas: 57635)
Test result: ok. 3 passed; 0 failed; finished in 955.64µs

Running 7 tests for contracts/test/unit/pcv/PCVGuardAdmin.t.sol:UnitTestPCVGuardAdmin
[PASS] testGrantPCVGuard() (gas: 99226)
[PASS] testGrantPCVGuardFailWhenGuardian() (gas: 17065)
[PASS] testGrantPCVGuardFailWhenNoRoles() (gas: 14577)
[PASS] testPCVGuardAdminRole() (gas: 10163)
[PASS] testRevokePCVGuardFailWhenNoRole() (gas: 19918)
[PASS] testRevokePCVGuardGovernor() (gas: 36082)
[PASS] testRevokePCVGuardGuardian() (gas: 38636)
Test result: ok. 7 passed; 0 failed; finished in 1.66ms

Running 11 tests for contracts/test/unit/pcv/ERC20HoldingPCVDeposit.t.sol:UnitTestERC20HoldingsP
[PASS] testDepositFailsOnPause() (gas: 40859)
[PASS] testDepositNoOp() (gas: 7591)
[PASS] testWithdrawAllFailsNonPCVController() (gas: 17439)
[PASS] testWithdrawAllSucceeds() (gas: 82588)
[PASS] testWithdrawERC20FailsNonPCVController() (gas: 16553)
[PASS] testWithdrawERC20Succeeds() (gas: 82261)
[PASS] testWithdrawEthFailsNonPCVController() (gas: 14410)
[PASS] testWithdrawEthSucceeds() (gas: 51551)
[PASS] testWithdrawFailsNonPCVController() (gas: 17497)
[PASS] testWithdrawSucceeds() (gas: 81625)
[PASS] testWrapEthFailsWhenNotOnMainnetOrArbitrum() (gas: 9021)
Test result: ok. 11 passed; 0 failed; finished in 22.95ms

Running 34 tests for contracts/test/unit/pcv/PCVGuardian.t.sol:UnitTestPCVGuardian
[PASS] testAddWhiteListAddress() (gas: 68365)
[PASS] testGovernorWithdrawAllERC20ToSafeAddress() (gas: 62208)
[PASS] testGovernorWithdrawERC20ToSafeAddress() (gas: 61455)
[PASS] testGuardianWithdrawAllERC20ToSafeAddress() (gas: 65460)
[PASS] testGuardianWithdrawAllToSafeAddress() (gas: 71328)
```

```
[PASS] testGuardianWithdrawERC20ToSafeAddress() (gas: 64622)
[PASS] testGuardianWithdrawToSafeAddress() (gas: 70016)
[PASS] testPCVGuardAdminRole() (gas: 10202)
[PASS] testPCVGuardWithdrawAllERC20ToSafeAddress() (gas: 68592)
[PASS] testPCVGuardWithdrawAllToSafeAddress() (gas: 74451)
[PASS] testPCVGuardWithdrawERC20ToSafeAddress() (gas: 67830)
[PASS] testPCVGuardWithdrawToSafeAddress() (gas: 73213)
[PASS] testPCVGuardianRoles() (gas: 13642)
[PASS] testPausedAfterWithdrawAllToSafeAddress() (gas: 105301)
[PASS] testPausedAfterWithdrawToSafeAddress() (gas: 104032)
[PASS] testRemoveWhiteListAddress() (gas: 25129)
[PASS] testWithdrawAllERC20ToSafeAddressFailWhenGuardRevokedGovernor() (gas: 48315)
[PASS] testWithdrawAllERC20ToSafeAddressFailWhenGuardRevokedGuardian() (gas: 50886)
[PASS] testWithdrawAllERC20ToSafeAddressFailWhenNoRole() (gas: 25445)
[PASS] testWithdrawAllERC20ToSafeAddressFailWhenNotWhitelist() (gas: 21962)
[PASS] testWithdrawAllToSafeAddress() (gas: 68108)
[PASS] testWithdrawAllToSafeAddressFailWhenGuardRevokedGovernor() (gas: 46547)
[PASS] testWithdrawAllToSafeAddressFailWhenGuardRevokedGuardian() (gas: 49153)
[PASS] testWithdrawAllToSafeAddressFailWhenNoRole() (gas: 23211)
[PASS] testWithdrawAlloSafeAddressFailWhenNotWhitelist() (gas: 19715)
[PASS] testWithdrawERC20ToSafeAddressFailWhenGuardRevokedGovernor() (gas: 50140)
[PASS] testWithdrawERC20ToSafeAddressFailWhenGuardRevokedGuardian() (gas: 52607)
[PASS] testWithdrawERC20ToSafeAddressFailWhenNoRole() (gas: 27607)
[PASS] testWithdrawERC20oSafeAddressFailWhenNotWhitelist() (gas: 24150)
[PASS] testWithdrawToSafeAddress() (gas: 66851)
[PASS] testWithdrawToSafeAddressFailWhenGuardRevokedGovernor() (gas: 48264)
[PASS] testWithdrawToSafeAddressFailWhenGuardRevokedGuardian() (gas: 50871)
[PASS] testWithdrawToSafeAddressFailWhenNoRole() (gas: 25402)
[PASS] testWithdrawToSafeAddressFailWhenNotWhitelist() (gas: 21838)
Test result: ok. 34 passed; 0 failed; finished in 33.85ms

Running 10 tests for contracts/test/unit/pcv/utils/ERC20AllocatorConnector.t.sol:UnitTestERC20Al
[PASS] testConnectAndRemoveNewDeposit() (gas: 1363694)
[PASS] testConnectNewDepositFailsTokenMismatch() (gas: 1344233)
[PASS] testConnectNewDepositFailsUnderlyingTokenMismatch() (gas: 1344278)
[PASS] testConnectNewDepositSkimToDripFrom() (gas: 1591079)
[PASS] testCreateDuplicateDepositFails() (gas: 24620)
[PASS] testCreateNewDepositFailsUnderlyingTokenMismatch() (gas: 2687951)
[PASS] testDripFailsToNonConnectedAddress(address) (runs: 256, μ: 15788, ~: 15788)
[PASS] testEditPSMTargetBalanceFailsPsmUnderlyingChanged() (gas: 180236)
[PASS] testSetTargetBalanceNonExistingPsmFails() (gas: 130986)
[PASS] testSkimFailsToNonConnectedAddress(address) (runs: 256, μ: 15701, ~: 15701)
Test result: ok. 10 passed; 0 failed; finished in 39.03ms

Running 12 tests for contracts/test/unit/utils/RateLimitedV2.t.sol:UnitTestRateLimitedV2
[PASS] testDepleteBuffer(uint128,uint16) (runs: 256, μ: 22136, ~: 25794)
[PASS] testDepleteBufferFailsWhenZeroBuffer() (gas: 20234)
[PASS] testDepleteThenReplenishBuffer(uint128,uint128,uint16) (runs: 256, μ: 28172, ~: 27111)
[PASS] testReplenishBuffer(uint128,uint16) (runs: 256, μ: 30682, ~: 31404)
[PASS] testReplenishWhenAtBufferCapHasNoEffect(uint128) (runs: 256, μ: 13537, ~: 13537)
[PASS] testSetBufferCapGovSucceeds() (gas: 30801)
```

```
[PASS] testSetBufferCapNonGovFails() (gas: 14344)
[PASS] testSetRateLimitPerSecondAboveMaxFails() (gas: 17059)
[PASS] testSetRateLimitPerSecondGovSucceeds() (gas: 29020)
[PASS] testSetRateLimitPerSecondNonGovFails() (gas: 14266)
[PASS] testSetRateLimitPerSecondSucceeds() (gas: 27052)
[PASS] testSetup() (gas: 13286)
Test result: ok. 12 passed; 0 failed; finished in 67.82ms

Running 45 tests for contracts/test/unit/pcv/utils/ERC20Allocator.t.sol:UnitTestERC20Allocator
[PASS] testAllConditionsFalseWhenPaused() (gas: 142927)
[PASS] testBufferDepletesAndReplenishesCorrectly() (gas: 261973)
[PASS] testBufferDepletesAndReplenishesCorrectlyMultipleDecimalNormalizedDeposits() (gas: 384961
[PASS] testBufferUpdatesCorrectly() (gas: 261566)
[PASS] testConnectDepositNonGovFails() (gas: 14567)
[PASS] testCreateDepositNonGovFails() (gas: 14728)
[PASS] testDeleteDepositGovSucceeds() (gas: 27065)
[PASS] testDeleteDepositNonGovFails() (gas: 14419)
[PASS] testDeletePSMGovSucceeds() (gas: 29270)
[PASS] testDeletePSMGovSucceedsDripFails() (gas: 36233)
[PASS] testDeletePSMGovSucceedsDripFailsDeleteDeposit() (gas: 35591)
[PASS] testDeletePSMGovSucceedsSkimFails() (gas: 36098)
[PASS] testDeletePSMGovSucceedsSkimFailsDeleteDeposit() (gas: 36628)
[PASS] testDeletePSMNonGovFails() (gas: 16548)
[PASS] testDoActionDripSucceedsWhenUnderFullTargetBalance(uint8) (runs: 256, μ: 217770, ~: 21777
[PASS] testDoActionFailsWhenUnderTargetWithoutPCVControllerRole() (gas: 116833)
[PASS] testDoActionNoOpOnNonWhitelistedPSM() (gas: 20223)
[PASS] testDoActionNoOpWhenUnderTargetWithoutPCVControllerRole() (gas: 29353)
[PASS] testDoActionSkimSucceedsWhenOverThresholdWithPCVControllerFuzz(uint128) (runs: 256, μ: 16
[PASS] testDripAndSkimFailsWhenPaused() (gas: 44873)
[PASS] testDripFailsOnNonWhitelistedPSM() (gas: 20189)
[PASS] testDripFailsWhenBufferExhausted() (gas: 261410)
[PASS] testDripFailsWhenBufferZero() (gas: 194307)
[PASS] testDripFailsWhenUnderFunded() (gas: 177840)
[PASS] testDripFailsWhenUnderTargetWithoutPCVControllerRole() (gas: 116701)
[PASS] testDripNoOpWhenUnderTargetWithoutPCVControllerRole() (gas: 50015)
[PASS] testDripSucceedsWhenBufferFiftyPercentDepleted() (gas: 243180)
[PASS] testDripSucceedsWhenBufferFiftyPercentDepletedDecimalsNormalized() (gas: 3639644)
[PASS] testDripSucceedsWhenBufferFiftyPercentDepletedDecimalsNormalizedNegative() (gas: 3639613)
[PASS] testDripSucceedsWhenOverThreshold() (gas: 217932)
[PASS] testDripSucceedsWhenOverThresholdAndPSMPartiallyFunded() (gas: 248718)
[PASS] testDripSucceedsWhenUnderFullTargetBalance(uint8) (runs: 256, μ: 198531, ~: 198531)
[PASS] testDripperFailsWhenUnderFunded() (gas: 137455)
[PASS] testGetAdjustedAmountDown(uint128) (runs: 256, μ: 6991, ~: 6991)
[PASS] testGetAdjustedAmountUp(uint128) (runs: 256, μ: 6683, ~: 6683)
[PASS] testPullSucceedsWhenOverThresholdWithPCVController() (gas: 219109)
[PASS] testSetup() (gas: 56236)
[PASS] testSkimFailsOnNonWhitelistedPSM() (gas: 20211)
[PASS] testSkimFailsWhenOverTargetWithoutPCVController() (gas: 93596)
[PASS] testSkimFailsWhenUnderFunded() (gas: 48287)
[PASS] testSkimSucceedsWhenOverThresholdWithPCVControllerFuzz(uint128) (runs: 256, μ: 175953, ~:
[PASS] testSweepGovSucceeds() (gas: 76589)
```

```
[PASS] testSweepNonGovFails() (gas: 16783)
[PASS] testTargetBalanceGovSucceeds() (gas: 34072)
[PASS] testeditPSMTargetBalanceNonGovFails() (gas: 14647)
Test result: ok. 45 passed; 0 failed; finished in 220.07ms

Running 10 tests for contracts/test/unit/oracle/VoltSystemOracle.t.sol:VoltSystemOracleUnitTest
[PASS] testCompoundBeforePeriodStartFails() (gas: 10763)
[PASS] testCompoundSucceedsAfterOnePeriod() (gas: 28455)
[PASS] testLERPPerDay() (gas: 198204)
[PASS] testLinearInterpolation() (gas: 30104)
[PASS] testLinearInterpolationFuzz(uint32) (runs: 256, μ: 21821, ~: 24128)
[PASS] testLinearInterpolationFuzzMultiplePeriods(uint32,uint8) (runs: 256, μ: 1344004, ~: 56383
[PASS] testLinearInterpolationUnderYearFuzzPeriods(uint24,uint8) (runs: 256, μ: 842437, ~: 33487
[PASS] testMultipleSequentialPeriodCompounds() (gas: 51930321)
[PASS] testNoLinearInterpolationBeforeStartTime(uint16) (runs: 256, μ: 13584, ~: 13584)
[PASS] testSetup() (gas: 12398)
Test result: ok. 10 passed; 0 failed; finished in 1.24s
```

License