arbitrary./execution

Sentiment

SENTIMENT CONTROLLER AND ORACLE SECURITY ASSESSMENT

July 27, 2022

Prepared For:

0xSnarks

Prepared By:

John Bird

Jasper Clark

Ian Bridges

Changelog:

June 3, 2022 *Initial report delivered*

July 27, 2022 *Final report delivered*

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report contains the results of Arbitrary Execution's security assessment of Sentiment's Controller and Oracle smart contracts. Sentiment is a permissionless, undercollateralized on-chain credit system that allows users to post assets as collateral in exchange for loans. The protocol uses smart contract Accounts to hold collateral and loans, Controller contracts to interact with external protocols, and Oracles to receive pricing data for assets.

Three Arbitrary Execution (AE) engineers conducted this review over a 2-week period, from May 16, 2022 to May 27, 2022. The audited commit for the Controller contracts was `38d76dc3c50887d0d5c4f096bf69c84c0c06c1d8` in the main branch of the `sentimentxyz/controller` repository. The audited commit for the Oracle contracts was `e5a267846a848ec89b78d30ea8b858ebee5eb8d2` in the main branch of the `sentimentxyz/oracle` repository. These repositories were private at the time of the engagement, so hyperlinks may not work for readers without access. All Solidity files in the two repos were in scope for this audit. The complete list of files is located in Appendix B.

Note: The core protocol was not in scope for this engagement. The `sentimentxyz/protocol` repository will be reviewed in an upcoming report.

The team performed a detailed, manual review of the codebase with a focus on core Controller and Oracle behavior, as well as Sentiment's integration with other protocols and price feeds. In light of the recent depeg of Terra (UST), the team also focused on oracle behavior with unexpected price data. In addition to manual review, the team used Slither for automated static analysis.

The assessment resulted in 15 findings ranging in severity from medium to note (informational). One medium severity finding in Sentiment's Chainlink oracle can impact Risk Engine calculations for issuing loans. One low severity issue was identified in transferring ownership of Oracle and Controller contracts. Another low severity finding was identified in the way the protocol's Controller Facade batches external calls. The remaining low severity findings involve edge cases and security best practices when interfacing with external contracts. The note severity findings contain observations regarding code hygiene, documentation, and other best practices.

FIX REVIEW UPDATE

FIX REVIEW PROCESS

After receiving fixes for the findings shared with Sentiment, the AE team performed a review of each fix. Each pull request was scrutinized to ensure that the core issue was addressed, and that no regressions were introduced with the fix. A summary of each fix review can be found in the *update* section for a finding. For findings that the Sentiment team chose not to address, the team's rationale is included in the update.

All major issues have been fixed or acknowledged by the Sentiment team. Sentiment decided to address L02 with off-chain monitoring. The full breakdown of fixes can be found in the Fixes Summary section.

AUDIT OBJECTIVES

AE had the following high-level goals for the engagement:

- Ensure that the Controller Facade and Oracle Facade contracts are implemented consistently with Sentiment's documentation
- Ensure that the individual oracle and controller implementations correctly interface with external contracts and price feeds
- Identify smart contract vulnerabilities
- Evaluate adherence to development best practices

AE also set specific goals around price oracle interactions:

- Identify ways in which valid, but anomalous, price data can impact lending risk calculations
- Evaluate error handling when fetching data from external price feeds
- Ensure decimal precision is handled properly

Controllers and Oracles will be covered in more detail in the System Overview section.

OBSERVATIONS

Sentiment's contracts are separated into three repositories. The oracle and controller code live in separate repositories, which are included as submodules in the core protocol. The protocol documentation can be found online and covers system components and major actors. The protocol has unit tests, but test coverage is missing for some of the controller implementations. The code would benefit from NatSpec documentation, and comments for constants in the controller contracts.

Sentiment Accounts can interact with external Decentralized Finance (DeFi) protocols, and this means the Sentiment protocol must interface with third-party APIs and price feeds. Most of the serious findings in this report involve assumptions surrounding external price feed behavior and the use of deprecated contract interfaces. Great care must be taken when interacting with external contracts, as interfaces and application behavior can change. Because of their role in determining Account risk factors, AE recommends careful review of every controller and oracle that is added to the protocol. AE does not recommend allowing third parties to integrate controllers or oracles without the Sentiment team's review or additional access controls.

SYSTEM OVERVIEW

USER CATEGORIES

The two user groups that interact with Sentiment are Lenders and Borrowers.

LENDERS

Lenders are responsible for providing liquidity to the protocol. This liquidity is lent out to borrowers as undercollateralized debt. By providing assets, lenders receive interest-bearing LTokens, which can be burned at a later point in time to retrieve the initial principal and accrued interest.

BORROWERS

Borrowers can create leveraged debt positions against deposited assets. They can use these positions to interact with other approved applications in the DeFi ecosystem. Borrowers interact with the Sentiment protocol through the use of Account smart contracts and the Account Manager.

SYSTEM COMPONENTS

Note: This engagement focused primarily on the Controller and Oracle contracts. Descriptions of other protocol components are included for context. A security review of the core protocol is forthcoming.

Sentiment users use Accounts to take loans and interact with external DeFi protocols. The Account Manager and Risk Engine use Controllers and Oracles to verify user transaction data and determine the health factor of an Account.

LTOKEN

For each asset accepted as collateral on Sentiment, one Lending Token (LToken) contract is deployed. These contracts allow users to deposit assets to the protocol and receive an interest-bearing token in return. LTokens are implemented using the ERC-4626 standard.

ACCOUNTS

Account smart contracts hold a borrower's collateral and loaned assets. An Account is a beacon proxy contract, that gets its implementation address from an upgradeable beacon contract. Because collateral and loaned assets are stored in the Account, the borrower does not have custody of the loaned assets. Accounts interact with other DeFi applications through the protocol Controller. Borrowers can withdraw assets from the Account to withdraw profits and reduce or close out their position. Accounts are created by the Account Factory, and users interact with Accounts via the Account Manager.

ACCOUNT MANAGER

The Account Manager is responsible for managing and initializing accounts for borrowers. It manages account states for externally owned accounts (EOAs) interacting with the protocol. It can also update Controller and Risk Engine addresses that are used by the protocol.

ACCOUNT FACTORY

The Account Factory is used by the Account Manager to create and deploy Account beacon proxies.

RISK ENGINE

The Risk Engine regularly computes and verifies the credit risk of an Account. It is called by the protocol when state mutating functions are performed on an Account, and is used to analyze the health factor of an Account.

REGISTRY

The Registry contract handles global storage for the Sentiment ecosystem. It tracks Account addresses and the corresponding EOAs, LToken addresses, and addresses of deployed protocol contracts.

CONTROLLER

Controllers enable Accounts to interact with external protocols. They are responsible for verifying outgoing calldata and enabling interactions from an Account to a particular target contract. For each external protocol that Sentiment interacts with, there must be a corresponding Controller contract. There are currently controllers for Aave, Compound, Curve, Uniswap, WETH, and Yearn.

CONTROLLER FACADE

The `ControllerFacade` smart contract is the common entrypoint to all Controller implementations. The facade maintains a mapping of tokens that Accounts can transact with, and a mapping of external smart contracts to Controller interfaces.

ORACLE

Oracles are used by the protocol to price various assets. For example, the Risk Engine will call oracles when determining risk for a loan to an Account.

ORACLE FACADE

The `OracleFacade` is the common entrypoint to all Oracle implementations. The protocol interacts with specific oracles by calling into the Oracle Facade. The facade maintains a mapping of addresses to oracle interfaces, and has functions for checking asset prices.

PRIVILEGED ROLES

In the context of the Controller and Oracle, the `admin` has special privileges.

ADMINISTRATOR

The Oracle Facade and Controller Facade contracts use a custom `Ownable` implementation that sets an `admin` address upon deployment. The `Ownable` contract uses this address in its `adminOnly` modifier, which is used to protect important functions in both repositories. Actions such as updating Controller and Oracle addresses, or allowing/disallowing tokens in a controller can only be done by the `admin`. After deployment, the `admin` can be updated by calling the `transferOwnership` function.

VULNERABILITY STATISTICS

Severity	Count
Critical	0
High	0
Medium	1
Low	4
Note	10

FIXES SUMMARY

Finding	Severity	Status
M01	Medium	Fixed in pull request #13
L01	Low	Fixed in pull request #19
L02	Low	Acknowledged
L03	Low	Fixed in pull requests #20 and #14
L04	Low	Fixed in pull request #21
N01	Note	Fixed in pull request #24
N02	Note	Acknowledged
N03	Note	Acknowledged
N04	Note	Fixed in pull requests #20 and #14
N05	Note	Fixed in pull requests #191 and #190
N06	Note	Fixed in pull requests #16 and #24
N07	Note	Fixed in pull requests #16 and #24
N08	Note	Acknowledged
N09	Note	Fixed in pull request #18
N10	Note	Fixed in pull request #15

FINDINGS

MEDIUM SEVERITY

[M01] USE OF DEPRECATED INTERFACE COULD ALLOW OVERLEVERAGED BORROWING

The `getPrice` function in `ChainlinkOracle.sol` uses `latestAnswer` from its `AggregatorV3Interface` contract to retrieve the price for a particular token:

```
    function getPrice(address token) external view override returns (uint) {
        return (
            (uint(feed[token].latestAnswer())*1e18)/
            uint(ethUsdPriceFeed.latestAnswer())
        );
```

Sentiment's `AggregratorV3Interface` is incorrectly named. The function definition for `latestAnswer` is located in Chainlink's `AggregatorInterface.sol`, and not in Chainlink's `AggregatorV3Interface.sol`.

More importantly, `latestAnswer` can return zero if no answer is reached, and must be handled by the caller. The `ChainlinkOracle` contract does not check if `latestAnswer` returns 0.

In the event that the call to `feed[token].latestAnswer` returns zero, the corresponding call to `_valueInWei` in RiskEngine.sol would also return 0. Utility functions like `isBorrowAllowed` use `_valueInWei` to determine if an account can borrow funds:

```
uint borrowValue = _valueInWei(token, amt); // ChainlinkOracle returns 0
return _isAccountHealthy(
    _getBalance(account) + borrowValue, // _getBalance(account) + 0
    _getBorrows(account) + borrowValue  // _getBorrows(account) + 0
);
```

If the price of a borrow is incorrectly calculated, users can borrow beyond the maximum leverage allowed by the protocol so long as their account is currently healthy.

In the event that `ethUsdPriceFeed.latestAnswer` returns zero, `getPrice` will revert.

Chainlink recommends using `latestRoundData` to retrieve the price of an asset. This function will raise an error if the price feed does not have data to report and will not return zero. The `latestRoundData` function also returns additional information that can be used to verify price data, such as the round ID and timestamp.

RECOMMENDATION

Consider using the `latestRoundData` function instead of `latestAnswer` to retrieve asset prices when interfacing with Chainlink.

If `latestAnswer` must be used, consider adding logic to handle return values of zero, and renaming `AggregratorV3Interface.sol` to `AggregatorInterface.sol` to avoid confusion.

UPDATE

Fixed in pull request #13 (commit hash `fff2ae20942cd96dfbf5c88a4576369dab7fd9e1`), as recommended.

LOW SEVERITY

[L01] INCORRECT BATCHING OF CONTROLLER CALLS

In `controllerFacade.sol`, `canCallBatch` allows Accounts to bundle multiple calls to an external protocol:

```
function canCallBatch(
    address[] calldata target,
    bool[] calldata useEth,
    bytes[] calldata data
) external view returns (bool, address[] memory, address[] memory) {
    uint lenMinusOne = target.length - 1;
    for(uint i = 0; i < lenMinusOne; ++i) {
        if(address(controllerFor[target[i]]) == address(0))
            return(false, new address[](0), new address[](0));
    }
    return controllerFor[target[lenMinusOne]].canCall(
        target[lenMinusOne],
        useEth[lenMinusOne],
        data[lenMinusOne]
    );
}
```

The current loop condition `i < lenMinusOne` (which is `i < target.length - 1`) skips the last element in the `target` array. For a `target.length` of 5, this loop will execute for elements 0, 1, 2, and 3, but not 4. Furthermore, the logic after the loop only calls `canCall` on the final entry in the batch. All other elements in `target`, `useEth`, and `data` are skipped.

This finding was assigned a severity of low, because the core protocol does not currently use `canCallBatch`.

RECOMMENDATION

Consider updating `canCallBatch` such that the loop condition is `i <= lenMinusOne`, and `canCall` is called for every element in `target`, `useEth`, and `data`.

UPDATE

Fixed in pull request #19 (commit hash `47b8a4a5b2c7b5614028f9c58b45ac45975185a8`). The unused `canCallBatch` function was removed.

[L02] TOKEN PRICE TIMESTAMPS NOT CHECKED

When requesting token price data from an external Chainlink data feed via the `getPrice` function, the Chainlink price oracle does not check the timestamp of when that price data was last generated:

```
function getPrice(address token) external view override returns (uint) {
    return (
        (uint(feed[token].latestAnswer())*1e18)/
        uint(ethUsdPriceFeed.latestAnswer())
    );
}
```

In the event that the external Chainlink data feed stops updating its pricing data, the price oracle will continue to use the last price available in the feed.

Since the Risk Engine uses token prices to determine the total balance of each account, using outdated pricing information would result in account balances not reflecting the actual value of the tokens held by the account.

In the event that token prices fall dramatically while the Chainlink data feed is not actively updating its pricing information, the Risk Engine would calculate the total balance of an account as higher than it actually is. When this occurs, an unhealthy account may be treated as healthy if the difference between the inaccurate account balance and the actual account balance is enough to push the account's balance-to-borrow ratio above the minimum threshold enforced by the Risk Engine. Account owners would then be able continue to borrow and withdraw tokens, even though their accounts are unhealthy.

This same issue was exploited after Chainlink paused its LUNA price feed, resulting in two DeFi protocols losing over 10 million dollars in collateral.

RECOMMENDATION

Consider using one of the following techniques to ensure that the the timestamp of the latest price data was generated within an acceptable timing window:

- When using `latestAnswer` to retrieve price data, make an additional call to `latestTimestamp`.
- When using `latestRoundData`, perform the check against the `updatedAt` value returned by `latestRoundData`.

UPDATE

Acknowledged. The Sentiment team plans to fix this using a different strategy, described as follows:

> *1. Add a check to assert that the L2 sequencer is active before fetching price from the feed*
>
> *2. Instead of doing an on chain check for heartbeat we will do this off chain using a bot or defender that can then alert the team via a tool like pager duty. a. In the future we will automate the bot to perform certain txs if such an alert is triggered*
>
> *3. In the future we also intend to move from maintaining Chainlink feeds on our end to using Chainlink's feed registry on Arbitrum, currently the Chainlink feed registry is only live on mainnet.*

The health check on the Arbitrum L2 Sequencer was added in pull request #15 (commit hash `c74eb757515ce369eea2b6a5775cd9c1b3180184`). While this check does prevent the Chainlink Oracle from retrieving price data while the Arbitrum L2 Sequencer is offline, it does not protect the protocol from using outdated price data when the Sequencer is online.

At the time of this report, the two other mitigation strategies (off-chain monitoring and the use of Chainlink's feed registry on Arbitrum) had not yet been designed or implemented. As such, they fell outside the scope of this audit.

[L03] LACK OF ZERO-ADDRESS CHECK WHEN TRANSFERRING OWNERSHIP

In the Controller's `Ownable.sol` and the Oracle's `Ownable.sol`, the `transferOwnership` function is used to update the administrator of the contract:

```
    function transferOwnership(address newAdmin) external virtual adminOnly {
        emit OwnershipTransferred(admin, newAdmin);
        admin = newAdmin;
    }
```

The function does not check that `newAdmin` is nonzero. If the current administrator transfers ownership to the zero address, contracts will have to be redeployed to gain access to functions marked with the `adminOnly` modifier.

It is less risky to use well-known library contracts for access control instead of creating your own. OpenZeppelin's Ownable contract contains similar functions and modifiers with proper safeguards in place.

RECOMMENDATION

Consider checking that `newAdmin` is nonzero, or prevent `transferOwnership` from setting `newAdmin` to an inaccessible address by splitting `transferOwnership` into a two step process, where the new administrator is required to claim ownership of the contract.

Furthermore, consider using OpenZeppelin's Ownable contract for access control.

UPDATE

Fixed in pull request #20 (commit hash `64093bda50a06eff570a16c445d44b5ed01d1845`) for the Controller, and in pull request #14 (commit hash `d8592eb8f065936c1f6b30325b7d0bbc9481dbf1`) for the Oracle. A check for the zero address has been added in `transferOwnership`.

[L04] POSSIBLE CTOKEN CETHER COLLISION

In lines 31 to 36 of the `CompoundController` contract, the `underlying` function is called on every cToken when handling redemptions:

```
try ICToken(target).underlying() returns (address tokenIn) {
    tokensIn[0] = tokenIn;
    return(true, tokensIn, tokensOut);
} catch {
    return(true, new address[](0), tokensOut);
}
```

To handle both redeeming `CErc20` and `CEther`, the code assumes that a failure case will always resolve to the Compound Wrapped Ether token. This could lead to unpredictable behavior as it is possible another Compound cToken could be added in the future that does not have the `underlying` function, such as custom bridge tokens. In this case, the bridge tokens would be incorrectly identified as `CEther` tokens.

RECOMMENDATION

Consider calling the `symbol` function on each cToken and checking it against the specific Compound Wrapped Ether symbol `cETH`.

UPDATE

Fixed in pull request #21 (commit hash `8fdbb7459d5c325d11fee8cc329e24f4926dd092`), as recommended.

NOTE SEVERITY

[N01] MISSING DOCUMENTATION FOR FUNCTION SIGNATURES

The constants that correspond to external function signatures in controller contracts lack documentation and are not standardized. For example, `AaveEthController.sol` defines a function signature `DEPOSIT = 0x474cf53d` that corresponds to the function name `depositETH`. In its current state, it is not immediately clear to a developer or user what external function these constants correspond to. Adding documentation will increase code readability and ease the addition of other function signatures by standardizing the process.

RECOMMENDATION

Consider documenting controller constants to include the corresponding function signature and external contract. For example the `AaveEthController.sol` could include the following documentation:

```
// depositETH(address, address, uint16)
//   from https://github.com/aave/protocol-
v2/blob/master/contracts/misc/WETHGateway.sol
```

UPDATE

Fixed in pull request #24 (commit hash `fd1c142fb8ab4f65245bdcb1a69d33f59375f69f`). Comments that include the corresponding function signature were added to the constants.

[N02] MISSING CHECKS FOR FIXED POINT DECIMAL COUNT

The Chainlink Oracle makes assumptions about the decimal places used by its external price feeds.

The price calculation in `getPrice` assumes that both calls to `latestAnswer` will return prices with the same decimal precision:

```
function getPrice(address token) external view override returns (uint) {
    return (
        (uint(feed[token].latestAnswer())*1e18)/
        uint(ethUsdPriceFeed.latestAnswer())
    );
}
```

If the calls to `feed[token].latestAnswer` and `ethUsdPriceFeed.latestAnswer` return values with different decimal precision, the value calculated in `getPrice` will be incorrect. Given the tentative list of price feeds that will be integrated into the protocol (ETH-USD, USDT-USD, BTC-USD, USDC-USD, and DAI-USD), both calls to `latestAnswer` in ChainlinkOracle.sol will return prices with 8 decimal places. However, there is still the risk that a price feed with a different `decimals` value can be added in the future.

RECOMMENDATION

Consider calling the `decimals` function on both the token and `ethUsdPriceFeed` and using the returned values to accurately calculate the price scaling in `getPrice`.

UPDATE

Acknowledged, and will not fix. Instead, in pull request #19 (commit hash `8ee62a053e4a5c113889d2a33843e61b7ec90e57`), the Sentiment team added the following line to the NatSpec comment for the `getPrice` function in `ChainlinkOracle.sol`:

```
/// @dev feed[token].latestRoundData should return price scaled by 8 decimals
```

Additionally, the Sentiment team provided the following response about their price feed approval process:

> *We will ensure that we only use Chainlink price feeds that return 8 decimals (i.e. USD-denominated feeds) to avoid incorrect scaling*

[N03] POTENTIAL LOSS OF PRECISION WHEN CALCULATING TOKEN PRICES

Both the `getPrice` and `cubicRoot` functions in `CurveTriCryptoOracle.sol` perform multiplication on the results of division operations, which are susceptible to truncation. For example:

```
function cubicRoot(uint x) internal pure returns (uint) {
    uint D = x / 1e18;
    for (uint i; i < 255;) {
        uint D_prev = D;
        D = D * (2e18 + x / D * 1e18 / D * 1e18 / D) / (3e18);
        uint diff = (D > D_prev) ? D - D_prev : D_prev - D;
        if (diff < 2 || diff * 1e18 < D) return D;
        unchecked { ++i; }
    }
    revert("Did Not Converge");
}
```

Multiplication of a truncated result exacerbates the imprecision introduced when the truncation occurred. Since this code pattern occurs in functions responsible for calculating token prices, there is a risk that the Curve price oracle returns inaccurate pricing information to the Risk Engine.

RECOMMENDATION

Consider using the FullMath library to retain full precision when mixing division and multiplication operations.

UPDATE

Acknowledged, and will not fix. Sentiment plans to re-deploy Curve's price data feed to Arbitrum and use that instead of maintaining their own version. However, that contract contains the same issue:

```
@pure
@internal
def cubic_root(x: uint256) -> uint256:
    # x is taken at base 1e36
    # result is at base 1e18
    # Will have convergence problems when ETH*BTC is cheaper than 0.01 squared
dollar
    # (for example, when BTC < $0.1 and ETH < $0.1)
    D: uint256 = x / 10**18
    for i in range(255):
        diff: uint256 = 0
        D_prev: uint256 = D
        D = D * (2 * 10**18 + x / D * 10**18 / D * 10**18 / D) / (3 * 10**18)
        if D > D_prev:
            diff = D - D_prev
        else:
            diff = D_prev - D
        if diff <= 1 or diff * 10**18 < D:
            return D
    raise "Did not converge"
```

At the time of writing this report, Sentiment considered the risk of precision loss acceptable, and offered the following statement:

> *We were looking into it internally since we wanted to be sure. But to the point, yes we're in consensus internally want to move ahead with it as-is and consider it to be an acceptable risk.*

[N04] MISSING CONSTRUCTOR ZERO-ADDRESS CHECKS

In the controller's `Ownable.sol` contract and the oracle's `Ownable.sol` contract, the constructor assigns the administrator of the contract:

```
    constructor(address _admin) {
        admin = _admin;
    }
```

The constructor does not check that `_admin` is nonzero. If the contract is deployed with a zero address, contracts will have to be redeployed to gain access to functions marked with the `adminOnly` modifier. This is currently mitigated by the way the `Ownable` abstract contract is used, because `msg.sender` is directly passed to the constructor. The additional safety check will prevent potential misuse.

It is less risky to use well-known library contracts for access control instead of creating your own. OpenZeppelin's Ownable contract contains safeguards to prevent misuse.

RECOMMENDATION

Consider checking that `_admin` is nonzero. Furthermore, consider using OpenZeppelin's Ownable contract for access control.

UPDATE

Fixed in pull request #20 (commit hash `64093bda50a06eff570a16c445d44b5ed01d1845`) for the Controller, and in pull request #14 (commit hash `d8592eb8f065936c1f6b30325b7d0bbc9481dbf1`) for the Oracle. A zero check has been added in each constructor.

[N05] INCOMPLETE TEST COVERAGE

While there are integration tests for most protocol controllers and oracles, the following controllers do not have corresponding tests:

- `AaveV3Controller.sol`
- `StableSwapController.sol`

Complete unit test coverage is critical for exercising code functionality, catching regressions, and identifiying edge cases that were not accounted for.

RECOMMENDATION

Consider adding tests for the `AaveV3Controller` and `StableSwapController`. In the future, consider following a test-driven development process when adding new features and components.

UPDATE

Fixed in pull request #191 (commit hash `c632d5a15c8c0db3f83b79cf1dd5340f8123dc03`) and pull request #190 (commit hash `d7fb25d06d4541eca5eb789686150180c60f440c`), as recommended.

[N06] INCONSISTENT SOLIDITY VERSIONS

Different Solidity compiler versions are used throughout the oracle and controller repositories. The following contracts mix versions:

- In `AaveV2Controller.sol`:
  - ^0.8.11 (controller/src/aave/AaveV2Controller.sol)
  - ^0.8.11 (controller/src/aave/IProtocolDataProvider.sol)
  - ^0.8.10 (controller/src/core/IController.sol)
  - ^0.8.10 (controller/src/core/IControllerFacade.sol)
- In `AaveV3Controller.sol`:
  - ^0.8.11 (controller/src/aave/AaveV3Controller.sol)
  - ^0.8.10 (controller/src/aave/IPoolV3.sol)
  - ^0.8.10 (controller/src/core/IController.sol)
  - ^0.8.10 (controller/src/core/IControllerFacade.sol)
- In `YearnController.sol`:
  - ^0.8.10 (controller/src/core/IController.sol)
  - ^0.8.11 (controller/src/yearn/YearnController.sol)
- In `AaveEthController.sol`:
  - ^0.8.11 (controller/src/aave/AaveEthController.sol)
  - ^0.8.10 (controller/src/core/IController.sol)
- In `ChainlinkOracle.sol`:
  - ^0.8.10 (oracle/src/chainlink/AggregatorV3Interface.sol)
  - ^0.8.10 (oracle/src/chainlink/ChainlinkOracle.sol)
  - ^0.8.10 (oracle/src/core/IOracle.sol)
  - ^0.8.0 (oracle/src/utils/Errors.sol)
  - ^0.8.0 (oracle/src/utils/Ownable.sol)
- In `YTokenOracle.sol`:
  - ^0.8.10 (oracle/src/core/IOracle.sol)
  - ^0.8.11 (oracle/src/yearn/YTokenOracle.sol)
- In `OracleFacade.sol`:
  - ^0.8.10 (oracle/src/core/IOracle.sol)
  - ^0.8.10 (oracle/src/core/OracleFacade.sol)
  - ^0.8.0 (oracle/src/utils/Errors.sol)
  - ^0.8.0 (oracle/src/utils/Ownable.sol)
- In `WETHOracle.sol`:
  - ^0.8.10 (oracle/src/core/IOracle.sol)
  - ^0.8.0 (oracle/src/weth/WETHOracle.sol)

Using different compiler pragma directives can leave contracts susceptible to different sets of compiler bugs.

RECOMMENDATION

Consider making the Solidity version consistent across all contracts within the project.

UPDATE

Fixed in pull request #16 (commit hash `4f5259bdf59e440a682c843f9a78e7e1430448c3`) for the Oracle, and pull request #24 (commit hash `fd1c142fb8ab4f65245bdcb1a69d33f59375f69f`) for the Controller. The compiler version selected was 0.8.15.

[N07] LACK OF NATSPEC DOCUMENTATION

The Oracle and Controller contracts lack docstrings. Incomplete documentation makes code more difficult to follow and can lead to errors if developers and users fill in their own assumptions in the absence of information. For example, the Curve protocol `StableSwapController` contract does not provide information on which Curve implementation it is based on, making it difficult to ensure proper API usage.

RECOMMENDATION

The Solidity documentation recommends NatSpec for all public interfaces (everything in the ABI). Consider implementing NatSpec-compliant docstrings for all public and external functions.

A good example is the OpenZeppelin ERC20 contract. It follows the NatSpec guidelines, and provides contract documentation that gives additional information about context and usage.

UPDATE

Fixed in pull request #16 (commit hash `4f5259bdf59e440a682c843f9a78e7e1430448c3`) for the Oracle, and in pull request #24 (commit hash `fd1c142fb8ab4f65245bdcb1a69d33f59375f69f`) for the Controller.

[N08] LACK OF ERROR HANDLING WITH ORACLE INTERACTION

Oracle implementations call out to external contracts for price information. The `CTokenOracle` calls the `exchangeRateStored` function in Compound's `CToken.sol` when checking the price of `CEther` or a `CErc20` token:

```
    function getCEtherPrice() internal view returns (uint) {
        // ...
        return ICToken(cETHER).exchangeRateStored().mul(1e8);
    }

    function getCErc20Price(ICToken cToken, address underlying) internal view
returns (uint) {
        // ...
        return cToken.exchangeRateStored().mul(1e8)
            .div(IERC20(underlying).decimals())
            .mul(oracle.getPrice(underlying));
    }
```

Calls to `exchangeRateStored` can revert on error with a message:

```
    function exchangeRateStored() public view returns (uint) {
        (MathError err, uint result) = exchangeRateStoredInternal();
        require(err == MathError.NO_ERROR, "exchangeRateStored:
exchangeRateStoredInternal failed");
        return result;
    }
```

This revert can be caught in a `try/catch` and handled by the Oracle implementation. This will enable the Oracle to revert with a custom error message, call a fallback price feed, or even retry the call. Chainlink's `getLatestRoundData` function can be surrounded by `try/catch` blocks in a similar fashion.

RECOMMENDATION

Consider surrounding external calls in Oracle implementations with `try/catch` blocks for more granular error handling.

UPDATE

Acknowledged, and will not fix. Sentiment's statement for this issue:

> *We will be passing the revert as-is from `cToken.exchangeRateStored` and looking out for events with off-chain monitoring.*

[N09] CONTRACTS USE FLOATING COMPILER VERSION PRAGMA

All controller and oracle contracts float their Solidity compiler versions (e.g. `pragma solidity ^0.8.10`).

Locking the compiler version prevents accidentally deploying the contracts with a different version than what was used for testing. The current pragma prevents contracts from being deployed with an outdated compiler version, but still allows contracts to be deployed with newer compiler versions that may have higher risks of undiscovered bugs.

It is best practice to deploy contracts with the same compiler version that is used during testing and development.

RECOMMENDATION

Consider locking the compiler pragma to the specific version of the Solidity compiler used during testing and development.

UPDATE

Fixed in pull request #18 (commit hash `3d83e57c67e7d307730b2ab2f196c3a7129bbb2a`), as recommended.

[N10] UNSAFE CASTING OF INT256 TO UINT256

The `getPrice` interface is used by the Risk Engine to calculate the balance of an account. As part of `getPrice`, the Chainlink price oracle (`ChainlinkOracle.sol`) calls `latestAnswer` to retrieve token price data from an external Chainlink data feed.

The `latestAnswer` function returns an int256, however the Chainlink price oracle casts this return value to an uint256. If `latestAnswer` were to return a negative value, the cast to uint256 would transform that value into a large, positive value.

If an account holds a token for which `latestAnswer` returns a negative value then `_getBalance` would calculate the account's balance as being greater than it actually is. In this case, if the discrepancy between the correct balance and the miscalculated balance inflates the account's balance-to-borrow ratio above the Risk Engine's acceptable threshold, then the account would be incorrectly marked as healthy in the Risk Engine:

```
function _isAccountHealthy(uint accountBalance, uint accountBorrows)
    internal
    pure
    returns (bool)
{
    return (accountBorrows == 0) ? true :
        (accountBalance.div(accountBorrows) > balanceToBorrowThreshold);
}
```

When this happens, an account owner would be able to bypass the risk mitigation checks enforced by the Risk Engine. Given the tentative list of Chainlink price feeds that will be integrated into the protocol (ETH-USD, USDT-USD, BTC-USD, USDC-USD, and DAI-USD), the likelihood of a negative value being returned is zero. However, properly handling negative values returned from Chainlink will mitigate this risk when integrating additional price feeds in the future.

RECOMMENDATION

Consider adding a check after each `latestAnswer` call to detect whenever a negative value is encountered and `revert` in such cases. If the RiskEngine needs to handle negative prices, then consider updating all contracts that interact with `getPrice` to use signed instead of unsigned integers.

UPDATE

Fixed in pull request #15 (commit hash `c74eb757515ce369eea2b6a5775cd9c1b3180184`). The `getPrice` function now checks whether `latestRoundData` returns a negative value and reverts if it does when the oracle retrieves both token and ETH price data.

APPENDIX

APPENDIX A: SEVERITY DEFINITIONS

Severity	Definition
Critical	This issue is straightforward to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
High	This issue is difficult to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
Medium	This issue is important to fix and puts a subset of users' data at risk and is possible to lead to moderate financial impact.
Low	This issue is not exploitable in a recurring basis and cannot have a significant impact on execution.
Note	This issue does not pose an immediate risk but is relevant to security best practices.

APPENDIX B: FILES IN SCOPE

CONTROLLER/SRC

```
./uniswap/UniV3Controller.sol
./uniswap/UniV2Controller.sol
./curve/CurveCryptoSwapController.sol
./curve/StableSwapController.sol
./aave/AaveV2Controller.sol
./aave/AaveV3Controller.sol
./core/ControllerFacade.sol
./compound/CompoundController.sol
./yearn/YearnController.sol
./aave/IPoolV3.sol
./uniswap/ISwapRouterV3.sol
./weth/WETHController.sol
./aave/AaveEthController.sol
./utils/Ownable.sol
./core/IControllerFacade.sol
./aave/IProtocolDataProvider.sol
./core/IController.sol
./uniswap/IUniV2Factory.sol
./curve/IStableSwapPool.sol
./compound/ICToken.sol
./utils/Errors.sol
```

ORACLE/SRC

```
./curve/CurveTriCryptoOracle.sol
./compound/CTokenOracle.sol
./chainlink/ChainlinkOracle.sol
./uniswap/UniV2LPOracle.sol
./yearn/YTokenOracle.sol
./core/OracleFacade.sol
```

```
./utils/Ownable.sol
./uniswap/IUniswapV2Pair.sol
./aave/ATokenOracle.sol
./chainlink/AggregatorV3Interface.sol
./compound/ICToken.sol
./utils/Errors.sol
./weth/WETHOracle.sol
./aave/IAToken.sol
./core/IOracle.sol
./utils/IERC20.sol
```