SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	FIAT DAO
Prepared by:	Sherlock
Lead Security Experts:	WatchPug, GimelSec, hickuphh3
Dates Audited:	Aug 1-19, 2022, Sep 19-25, 2022, Oct 3-5, 2022
Prepared on:	Nov 9, 2022

Introduction

The FIAT protocol allows users to mint a single ERC-20 token, $FIAT, against a universe of accepted fixed income asset collateral. By providing users with fungible liquidity for the duration of their fixed term deposits in other protocol, FIAT reduces the opportunity cost of underwriting DeFi debt and aggregates secondary liquidity around a singular focal point.

Scope

First Commit Branch:

- fiat/main
- delphi-ii/main
- vaults/main
- actions/main

Commit:

- 9f9bd8dec4f27ce2ccf84d22683b1b313e899f62
- 35819264b12d5e204cb7ff18d3317e4aba7c7178
- d80d9411ee760da01f17f3019f8042a61f505722
- ba925673106997caae5da5a2179208b031995e9d

Contracts:

- delphi-v2/src/OptimisticOracle.sol
- delphi-v2/src/validators/utils/MerklePatriciaProofVerifier.sol
- delphi-v2/src/validators/utils/RLPReader.sol
- delphi-v2/src/validators/utils/StateProofVerifier.sol
- delphi-v2/src/validators/ChainlinkValidator.sol
- fiat/src/Aer.sol
- fiat/src/Limes.sol
- fiat/src/Flash.sol
- fiat/src/auctions/NoLossCollateralAuction.sol
- fiat/src/Publican.sol
- fiat/src/Collybus.sol
- fiat/src/Codex.sol

- fiat/src/FIAT.sol
- fiat/src/Moneta.sol
- fiat/src/Tenebrae.sol
- fiat/src/auctions/CollateralAuction.sol
- fiat/src/auctions/DebtAuction.sol
- fiat/src/auctions/PriceCalculator.sol
- fiat/src/auctions/SurplusAuction.sol
- fiat/src/utils/Guarded.sol
- fiat/src/utils/Math.sol
- vaults/src/VaultEPT.sol
- vaults/src/VaultFC.sol
- vaults/src/VaultFY.sol
- vaults/src/Vault.sol
- vaults/src/VaultFactory.sol
- actions/src/vault/VaultFYActions.sol
- actions/src/auction/NoLossCollateralAuctionActions.sol
- actions/src/vault/VaultFCActions.sol
- actions/src/lever/Lever20Actions.sol
- actions/src/lever/LeverActions.sol
- actions/src/lever/LeverEPTActions.sol
- actions/src/vault/VaultEPTActions.sol
- actions/src/helper/ConvergentCurvePoolHelper.sol
- actions/src/vault/Vault1155Actions.sol
- actions/src/vault/Vault20Actions.sol
- actions/src/vault/VaultActions.sol

Second Commit
Branch: feat/refactor-review
Commit: 181d85bd82ea1e5d468767e5368111734c39b908
Contracts:

- delphi-v2/src/OptimisticOracle.sol
- delphi-v2/src/validators/ChainlinkValidator.sol
- delphi-v2/src/OptimisticChainlinkOracle.sol

Protocol Attributes

Test coverage: Very good
Quality of tests: High

Language: Solidity
Assembly usage: No
Solc version: 0.8.0

Blockchain: Ethereum
L2s: None

Findings

Each issue has an assigned severity:

- Informational issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- Low issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
38	14	10	7

Issue H-1 Attacker can `dispute()` with malicious `data` and claim bonds from honest proposers

Summary

The `data` provided by the caller of `dispute()` can be malicious, which allows the attacker to dispute and claim bonds from honest proposers.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

The check

```
bool nonceIsValid = dataRoundTimestamp == roundTimestamp &&
   (nonce >> 64) ==
   (keccak256(abi.encode(dataRoundId, uint64(roundTimestamp))) >> 64);
```

attempts to ensure that the data matches the nonce, but erroneously translates a failing match as an invalid nonce. This isn't necessarily true as it might be the data that is malicious instead.

Impact

The caller of `dispute()` can use malicious `data` to dispute honest proposals and claim the bonds.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L176-L212

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticChainlinkOracle.sol#L143-L205

Tool used

Manual Review

Recommendation

The pre-image check should be converted to a reverting conditional check. `nonceIsValid` can subsequently be renamed to `timestampIsValid`.

```
// Verify that `data` is the pre-image of the hash encoded within `nonce`
if (
   (nonce >> 64) !=
   (keccak256(abi.encode(dataRoundId, uint64(roundTimestamp))) >> 64)
) revert OptimisticChainlinkOracle__validate_invalidData();
```

```
- bool nonceIsValid = dataRoundTimestamp == roundTimestamp &&
-    (nonce >> 64) ==
-    (keccak256(abi.encode(dataRoundId, uint64(roundTimestamp))) >> 64);
+ bool timestampIsValid = dataRoundTimestamp == roundTimestamp;
```

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/24. We refactored the `validate()` function by moving all time-related and data checks to shift(). The validate() method ensures that the **roundId** is correct(leads to a valid chainlink round) and that the proposed value is the same as the computed one.

Sherlock

Do we still need the `data` parameter in `dispute()`, seems like it's no longer used, can we get rid of it?

Team

The data param in dispute is not used by the chainlink oracle but other implementations(like the proof oracles) will use it. We kept it because of that reason.

Sherlock

In that case, maybe we can remove this line so it's more clear that data is not needed for Chainlink: https://github.com/fiatdao/delphi-v2/blob/608ff90d260a92d39140af4f7b73f78642c8886c/src/OptimisticChainlinkOracle.sol#L172

Team

Removed the `data` param from the `validate()` function.

Sherlock

Confirmed.

Issue H-2 Stale price can be disputed but still accepted as a valid price

Summary

The `shift()` function has no validation that stops a proposer from creating a new proposal with valid but outdated price data.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

If an attacker submitted a proposal with the price from a long time ago, say 10 days, this proposal can and will be disputed.

However, `dispute()->validate()` won't refetch the new price data as long as `nonceIsValid` (L183-198), so that the price from 10 days ago will be passed to `_settleDispute()` as the `validValue` (OptimisticChainlinkOracle.sol#L191, OptimisticOracle.sol#L188,207).

Impact

Even though the `dispute()` won't push the stale price to the Collybus immediately, instead, it creates a new proposal in the name of the `OptimisticChainlinkOracle` contract itself as the proposer. But since this proposal is not bonded, it may not get disputed, so the stale price can likely be shifted to the collybus soon after the dispute.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticChainlinkOracle.sol#L143-L205

Tool used

Manual Review

Recommendation

Considering the fact that the `nonce` is computed on-chain in `shift()` (L156). It can and should make sure the new proposal is within the `proposeWindow`.

Thus, we can get rid of the `proposeWindow` check in the `validate()` function.

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/24 We moved all time-related checks in shift. The `proposeWindow` was removed in favor of checking that the proposed value is newer than the previous proposal. This is preferred because it gives us a generic way of handling different token types with different update rates. The side-effect is that now price updates can start to lag behind if for example the dispute window is let's say 6 hours and the chainlink feed is updated every hour, then a possible attack vector would be to push the next computed(compared to the current proposed value) round instead of the latest or a newer one and in time this can lead to stale prices being pushed or that an attacker has a big pool of chainlink rounds to choose from. In order to avoid this problem we will have a keeper running that ensures the price updates do not fall behind. If we detect any lag the keeper will execute a 'push() which will update to the latest chainlink value().

The reason we do not check the round timestamp in shift is that it makes the optimistic proposal system obsolete by making shifts more expensive that pushes.

Sherlock

Fixed

Issue H-3 Surplus auction cannot be cancelled

Summary

Attempts to cancel surplus auctions will likely fail because approval is required, but not given.

Vulnerability Detail

Instead of performing a conventional `transfer()`, the token's `transferFrom()` function is called:

```
token.transferFrom(address(this), auctions[auctionId].recipient,
↪  auctions[auctionId].bid);
```

However, approval needs to be given to the caller, even if the caller is the token sender. This is the case for OpenZeppelin's ERC20 implementation (which FDT inherits). A snippet of the `transferFrom()` function is given below.

```
_transfer(sender, recipient, amount);
uint256 currentAllowance = _allowances[sender][_msgSender()];
require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
```

Because zero allowance has been given, the transaction will revert.

Impact

Surplus auctions cannot be cancelled.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/SurplusAuction.sol#L192

Tool used

Manual Review

Recommendation

Change to `transfer()` or OpenZeppelin's `safeTransfer()` methods.

```
token.transfer(auctions[auctionId].recipient, auctions[auctionId].bid);
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue H-4 `SurplusAuction.solcloseAuction()` will revert as FDT cant transfer to the zero address

Summary

`SurplusAuction.solcloseAuction()` will revert at L173 as FDT can not transfer to the zero address.

Vulnerability Detail

See L173, `token.transfer()` with `address(0)` as `to` is not allowed.

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/auctions/SurplusAuction.sol#L165-L175

```
function closeAuction(uint256 auctionId) external override {
    if (live == 0) revert SurplusAuction__closeAuction_notLive();
    if (
        !(auctions[auctionId].bidExpiry != 0 &&
            (auctions[auctionId].bidExpiry < block.timestamp ||
                auctions[auctionId].auctionExpiry < block.timestamp))
    ) revert SurplusAuction__closeAuction_notFinished();
    codex.transferCredit(address(this), auctions[auctionId].recipient,
↪   auctions[auctionId].creditToSell);
    token.transfer(address(0), auctions[auctionId].bid);
    delete auctions[auctionId];
}
```

Impact

`SurplusAuction.solcloseAuction()` always reverts.

Code Snippet

Tool used

Manual Review

Recommendation

`token` should be sent to the `governance` address instead:

```
function closeAuction(uint256 auctionId) external override {
    if (live == 0) revert SurplusAuction__closeAuction_notLive();
    if (
        !(auctions[auctionId].bidExpiry != 0 &&
```

```
            (auctions[auctionId].bidExpiry < block.timestamp ||
                auctions[auctionId].auctionExpiry < block.timestamp))
    ) revert SurplusAuction__closeAuction_notFinished();
    codex.transferCredit(address(this), auctions[auctionId].recipient,
↪   auctions[auctionId].creditToSell);
    token.transfer(governance, auctions[auctionId].bid);
    delete auctions[auctionId];
}
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue H-5 `nonce` could be malicious in `OptimisticOracle.shift`

Summary

Proposers can use malicious `nonce` in `OptimisticeOracle.shift`, since there is no check for `nonce`.

Vulnerability Detail

`nonce` is provided by the proposer. Thus, `roundId` and `roundTimestamp` can be malicious.

1. A malicious `roundTimestamp` can skip disputeWindow due to this issue https://github.com/SherlockAudit/fiat-dao/issues/31
2. A malicious `roundId` can cheat `ChainlinkValidator.validate`, e.g. Use old `roundId`, then `ChainlinkValidator.validate` can accept old `value`(which could be a bad value in the currect round).

Impact

Proposers can use malicious `nonce` to set bad values. Bad values will be pushed to FIAT. It could lead to a big disaster in FIAT protocol.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L187-L237

Tool used

Manual Review

Recommendation

Add a check for `nonce`. Or let `OptimisticOracle` provide `roundTimestamp` and `roundId` instead of proposers.

Team

Added a check in `ChainlinkValidator.validate()` that ensures the **timestamp** specified in the **nonce** is the same as the chainlink data round `updatedAt` timestamp. The checks in `OptimisticOracle.shift()` will ensure the nonce timestamp is passing our window checks and the `validate()` function should catch cases where a **roundId**

was used with a different timestamp. I think this covers cases where old//malicious **roundIds** or timestamps are set via shift(). PR: https://github.com/fiatdao/delphi-v2/pull/5

Sherlock

```
// Check that the feed timestamp matches the nonce
if (roundTimestamp != decodeRoundTimestamp(nonce)) {
    revert ChainlinkValidator__validate_invalidNonce();
}
```

revert means that dispute fail. Thus if roundId is invalid, no one can dispute the proposal.

Maybe, there can be a `lastRoundId` or `latestRoundId` to prevent that users use old roundId

Team

You are correct, we should not revert in case of an invalid nonce, the dispute should be successful in that case. Also we should not revert on the `disputeWindow` if the `nonce` <-> `roundTimestamp` relation was not validated. I will update the flow to first validate the feed timestamp and only after that can it revert on dispute window passed. Also if the feed timestamp is not validated the dispute will always return success even if the proposed value somehow is correct(same as the computed one). This was a good catch, we changed the dispute to revert and obviously, I didn't consider all the implications.

So the global flow would be that the OptimisticOracle will validate that the nonce used for the dispute is the same as the nonce used in the proposal and if the validated nonce contains invalid chainlink data for eg invalid round timestamp then the dispute will be successful.

Updated the PR.

Sherlock

There is one minor issue still remains: when the original nonce is invalid, the `dispute()` should not continue using the wrong `nonce` in `_settleDispute()`.

Maybe consider allowing the caller to specify another, correct nonce in this case?

https://github.com/fiatdao/delphi-v2/blob/96570ef83f978616ef4a4a6057f69364a1206c57/src/OptimisticOracle.sol#L289-L321

```
function dispute(
        bytes32 rateId,
```


```
        address proposer,
        address receiver,
        uint256 value,
        bytes32 nonce
    ) external {
        RateConfig memory rateConfig = rateConfigs[rateId];
        if (rateConfig.validator == address(0))
            revert OptimisticOracle__dispute_rateConfigNotSet();

        // Validate the proposed value by fetching it from the corresponding
↪   Chainlink feed
        (bool proposalIsValid, uint256 verifiedValue) = IChainlinkValidator(
            address(rateConfig.validator)
        ).validate(
                value,
                address(uint160(uint256(rateId))), // RateId encodes the address
↪   of the token
                nonce // Nonce encoded the roundId and the roundTimestamp
            );

        // Proposal has to be invalid
        if (proposalIsValid) revert OptimisticOracle__dispute_invalidDispute();

        _settleDispute(
            rateId,
            proposer,
            receiver,
            value,
            verifiedValue,
            nonce,
            address(rateConfig.validator)
        );
    }
```

https://github.com/fiatdao/delphi-v2/blob/96570ef83f978616ef4a4a6057f69364a1206c57/src/OptimisticOracle.sol#L331-L358

```
function _settleDispute(
    bytes32 rateId,
    address proposer,
    address receiver,
    uint256 value,
    uint256 computedValue,
    bytes32 nonce,
    address validator
) private {
    if (proposer == validator) {
```

```
        revert OptimisticOracle__settleDispute_alreadyDisputed();
    }

    // Verify the proposal data
    if (
        proposals[rateId] !=
        computeProposalId(rateId, proposer, value, uint256(nonce))
    ) {
        revert OptimisticOracle__settleDispute_unknownProposal();
    }

    // Overwrite the proposal with the value computed by the Validator
    proposals[rateId] = computeProposalId(
        rateId,
        address(validator),
        computedValue,
        uint256(nonce)
    );
```

Team

Updated the PR. Now the nonce will be computed by the validator. Mainly it will pack the propose timestamp with the provided validator data and the resulting nonce will be used to check the dispute / propose windows. This means that we can check the dispute window when we are attempting to make a new shift and the freshness of the provided data is now checked only in disputes(besides the regular checks). The validator created nonce contains the **roundId**, **roundTimestamp**, and the **propose-Timestamp** packed together.

We still need to update the test contracts to better reflect the changes and also add tests for some issues found during the audit (like the double shift).

Because of the issuer regarding malicious nonce double shifting we had to revisit how we create and use the nonce in the validation process. The PR for this issue has been closed and we are working on a separate PR that changes the nonce management and reorganizes the contract architecture a bit. We discussed this issue on discord in the sherlock channel and the current plan is for us to finalize this PR and we will have a separate review for delphi.

Sherlock

Team plans to address this issue further in the refactor-review update from Oct 3rd to Oct 5th.

Issue H-6 `OptimisticOracle.unbond` can be tricked by malicious users.

Summary

Malicious users can bypass the check of `OptimisticOracle.unbond`.

Vulnerability Detail

`OptimisticOracle.unbond` checks whether the current proposal has not been made by msg.sender.

```
if (
    proposals[rateId] == proposalId &&
    rateConfig.validator != address(0) &&
    IValidator(rateConfig.validator).canDispute(nonce)
) {
    revert OptimisticOracle__unbond_isProposing();
}
```

However, `nonce` and `value` is given by `msg.sender`. So `proposalId` is determined by `msg.sender`.

```
function unbond(
    bytes32 rateId,
    uint256 value,
    bytes32 nonce,
    address receiver
) public {
    bytes32 proposalId = computeProposalId(
        rateId,
        msg.sender,
        value,
        uint256(nonce)
    );

    // Current proposal has not been made by `msg.sender` or it has passed
↪   `disputeWindow`
    // or the rateConfig has been unset
    RateConfig memory rateConfig = rateConfigs[rateId];
    if (
        proposals[rateId] == proposalId &&
        rateConfig.validator != address(0) &&
        IValidator(rateConfig.validator).canDispute(nonce)
    ) {
        revert OptimisticOracle__unbond_isProposing();
```

```
    }
    ...
}
```

Thus, the check of `proposalId` can be easily bypassed.

Impact

Proposers can easily bypass the check in `OptimisticOracle.unbond` and successfully take back their bond tokens when they are in the dispute window.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L463-L495

Tool used

Manual Review

Recommendation

Add a mapping for `proposer` and `rateId`: `proposer[rateId]` (Should be set in `OptimisticOracle.shift`).

Check `proposer[rateId]` instead of `proposals[rateId]`.

```
    function unbond(
        bytes32 rateId,
-       uint256 value,
-       bytes32 nonce,
        address receiver
    ) public {
-       bytes32 proposalId = computeProposalId(
-           rateId,
-           msg.sender,
-           value,
-           uint256(nonce)
-       );

        // Current proposal has not been made by `msg.sender` or it has passed
 ↪  `disputeWindow`
        // or the rateConfig has been unset
        RateConfig memory rateConfig = rateConfigs[rateId];
        if (
+           proposer[rateId] == msg.sender &&
            rateConfig.validator != address(0) &&
```

```
            IValidator(rateConfig.validator).canDispute(nonce)
        ) {
            revert OptimisticOracle__unbond_isProposing();
        }
        ...
    }
```

Team

Fixed in PR: https://github.com/fiatdao/delphi-v2/pull/6

Sherlock

Fixed.

Issue H-7 Incorrect translation of require checks to revert with custom error checks

Summary

There are a number of instances where the require checks in the original libraries were incorrectly translated to revert conditions.

Vulnerability Detail

One of the changes to the utils libraries from the referenced ones were the use of custom errors instead of require statements. This means that conditions and checks have to be inverted. For instance, `require(x==0)` is to be replaced with `if(x!=0)revertCustomError()`.

One of these conditions was inverted incorrectly. The original statement `require(x.length>0);` should have been converted to `if(x.length==0)revertCustomError()`, but was incorrectly converted to `if(x.length!=0)revertCustomError()` instead.

Impact

Broken functionality.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/MerklePatriciaProofVerifier.sol#L233-L234 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/MerklePatriciaProofVerifier.sol#L262-L263 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/RLPReader.sol#L269

Tool used

Manual Review

Recommendation

```
- if (compact.length != 0)
+ if (compact.length == 0)
   revert MerklePatriciaProofVerifier__decodeNibbles_compactIsZero();

- if (compact.length != 0)
+ if (compact.length == 0)
```

```
    revert
↪   MerklePatriciaProofVerifier__merklePatriciaCompactDecode_compactIsZero();

- if (item.len != 0) revert RLPReader__toBytes_invalidLen();
+ if (item.len == 0) revert RLPReader__toBytes_invalidLen();
```

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/7

Sherlock

Fixed.

Issue M-1 Proposer can submit malicious `data` that cannot be disputed

Summary

A proposer can submit malicious `data` that forces dispute attempts to be reverted, thus preventing disputes (and is therefore able to submit a malicious value unless `push()` is called).

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

There are 2 ways in which a malicious proposer can force a dispute transaction to revert.

1. ABI Decoding

The data cannot be decoded to `uint80dataRoundId` and `uint64dataRoundTimestamp`. (Eg. data = `0x`).

```
// Retrieve the round data from Chainlink
(uint80 dataRoundId, uint64 dataRoundTimestamp) = abi.decode(
    data,
    (uint80, uint64)
);
```

2. Malicious `dataRoundId`

The `dataRoundId` used is out of range (eg. a future `dataRoundId` or low `dataRoundId`).

```
(
   ,
   int256 roundValue,
   ,
   uint256 roundTimestamp,

) = AggregatorV3Interface(feed).getRoundData(dataRoundId);
```

Impact

Proposals with malicious `dataRoundId` cannot be disputed. While this can partially be mitigated by `push()`, which computes and pushes a value directly, the proposer will

nevertheless be able to continue submitting these malicious proposals as his bond cannot be removed, causing DoS of the `shift()` and `dispute()` functionality.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticChainlinkOracle.sol#L164-L175

Tool used

Manual Review

Recommendation

Wrap the decode and `getRoundData()` calls in a try-catch block; the validity of the proposal should be set to `false` in the catch block.

Team

Fixed in: https://github.com/fiatdao/delphi-v2/pull/24 Data is now decoded in `shift()` where reverting is ok and the chainlink call is wrapped in a try-catch where a revert will lead to a successful dispute.

Sherlock

Fixed.

Issue M-2 `_settleDispute()` does not block the proposer from further bonding

Summary

`blockCaller()` in `_settleDispute()` is supposed to block the proposer from further bonding, but since it only removes the root access from the proposer, which they don't have, it won't block it.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

```
function blockCaller(bytes32 sig, address who) public override callerIsRoot {
    _canCall[sig][who] = false;
    emit BlockCaller(sig, who);
}

/// @notice Returns if `who` can call `sig`
/// @param sig Method signature (4Byte)
/// @param who Address of who should be able to call `sig`
function canCall(bytes32 sig, address who) public view override returns (bool) {
    return (_canCall[sig][who] || _canCall[ANY_SIG][who] ||
↪   _canCall[sig][ANY_CALLER]);
}
```

OptimisticOracle.solL280 calls the internal function `blockCaller()`, removes the root access from the `proposer`, which the `proposer` should not have in the first place. Therefore, it's essentially a no-op.

Impact

The proposer of a malicious proposal can bond again and continue to submit new proposals.

Code Snippet

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/utils/Guarded.sol#L62-L72

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L247-L283

Tool used

Manual Review

Recommendation

Consider changing the `ANY_SIG` in `blockCaller()` to the method signature of `bond()`.

Team

Fixed in: https://github.com/fiatdao/delphi-v2/pull/24 Changed the block caller to use the bond signature.

Sherlock

Not fully fixed, missing out on blocking the other bond() function.

Team

True, we don't block for the delegate `bond(address,rates)` function but that will be callable only by the oracle owner(for keeper setup). Regular proposers will only have permission to call `bond(rateIds)via`allowProposer()‘.

Sherlock

Acknowledged: `bond(bytes32[])` is blocked, and while `bond(address,bytes32[])` isn't, it is expected to be restricted to only the oracle owner.

Issue M-3 `dispute()` can only be called by root

Summary

`dispute()->_settleDispute()->blockCaller()` requires root access, thus `dispute()` requires root access.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

The internal function `blockCaller()` can only be called by root because of the `callerIsRoot` modifier:

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/utils/Guarded.sol#L62-L72

```
function blockCaller(bytes32 sig, address who) public override callerIsRoot {
    _canCall[sig][who] = false;
    emit BlockCaller(sig, who);
}
```

OptimisticOracle.solL280 calls the internal function `blockCaller()`, which makes the whole `dispute()` function only callable by root.

Impact

`dispute()` is no longer a permissionless method.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L247-L283

Tool used

Manual Review

Recommendation

Consider changing to `this.blockCaller()` and grant the contract itself root access.

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/24 An internal `_blockCaller()` method was added and is used in `_settleDispute()`

Sherlock

Fixed.

Issue M-4 `rate` should be updated before `modifyCollateralAndDebt()`, thus repayment can be made at a lower rate than expected

Summary

If the user is not calling through `VaultActions`, `vaults[vault].rate` will not be updated before `modifyCollateralAndDebt()`, thus repayment can be made at a lower rate than expected.

Vulnerability Detail

At L124 in `VaultActions.sol`, `publican.collect(vault)` will be called to update `vaults[vault].rate` of codex with `codex.modifyRate()`.

However, the user may not repay their debt using `VaultActions` but interact with the codex contract directly.

By doing so, `codex.modifyCollateralAndDebt()` will use the old rate, which should be updated with `publican.collect(vault)`.

Impact

If `vaults[vault].rate` is not updated by others, then the user can avoid the interests since the last updated time.

Code Snippet

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/Codex.sol#L297-L352

https://github.com/fiatdao/actions/blob/e9b5148f75349884a984be4f458fb2ae9e690599/src/vault/VaultActions.sol#L113-L171

Tool used

Manual Review

Recommendation

The same problem also exists in MakerDAO.

Seems like MakerDAO has realized this issue and set up a `keeper` bot to call `drip()` from time to time and update the `rate`: https://etherscan.io/address/0x0a51500250d1f6e2612a5d14d2094b5573635774

Team

This is expected and should not be labeled as 'high'. We have a service which collects the interest every 3 days as well on Gelato. In the worst case the protocol may loose out on a tiny amount of due interest, though on the other hand we don't have to resort for a more explicit interest accounting mechanism - which saves the users gas in the end. When Maker designed this mechanism they were aware of the tradeoffs. Acknowledged.

Sherlock

Acknowledged.

Issue M-5 Malicious proposers can frontrun `dispute()` and get back their bond to minimize the penalty

Summary

`dispute()` is a permissionless function that anyone can call to claim all the bonds from an invalid proposal. This allows the malicious proposers to claw back their bond by front-running `dispute()` and minimizing the penalty.

Vulnerability Detail

When a malicious proposal front run `dispute()` on their own invalid proposal, the real cost is the gas cost of the dispute transaction.

A very sophisticated attacker can deliberately create a few invalid proposals, frontrun others' `dispute()` transactions, and waste their gas.

As a result, the other dispute bots will be pushed out as they often get front run, resulting in those bots wasting gas costs on failed transactions.

After a while, the attacker can casually add an invalid proposal, and there is a chance that no other bots will `dispute()`, and even if they do, the attacker will just frontrun them again, with very minimal cost.

Impact

Even if the `bondSize` is very large, say $5000, it becomes insufficient to prevent malicious proposals. As the front run cost can be as low as $50, and the potential gains with a malicious price proposal can be >500k.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/96570ef83f978616ef4a4a6057f69364a1206c57/src/OptimisticOracle.sol#L289-L321

https://github.com/fiatdao/delphi-v2/blob/96570ef83f978616ef4a4a6057f69364a1206c57/src/OptimisticOracle.sol#L331-L366

https://github.com/fiatdao/delphi-v2/blob/96570ef83f978616ef4a4a6057f69364a1206c57/src/OptimisticOracle.sol#L508-L538

Tool used

Manual Review

Recommendation

1. Instead of sending all the`bondSize` to the `receiver` of `dispute()`, consider sending only a portion of the bond, say 50%, and use the other 50% as rewards for the honest proposers; then the real cost for a malicious proposer would be at least 50% of the `bondSize`.
2. Consider optimizing the gas cost for `dispute()` a proposal that is already been disputed, the current implementation is quite expensive.

Specifically, consider moving L345-L350 to before L301, so that it reverts earlier and saves gas for the bots:

https://github.com/fiatdao/delphi-v2/blob/96570ef83f978616ef4a4a6057f69364a1206c57/src/OptimisticOracle.sol#L345-L350

```
if (
            proposals[rateId] !=
            computeProposalId(rateId, proposer, value, uint256(nonce))
        ) {
            revert OptimisticOracle__settleDispute_unknownProposal();
        }
```

Team

We are going to restrict the `bond` method and revoke rights to call `bond` from slashed proposers. The only attack vector that remains is a 1 hour long block withholding attack. Though this is only solvable by having more complex range bound checks in Collybus. Which we might look into in the future. I'll leave the bondSize payout as is.

An attacker can attempt to use another address to bond, but it would have to be whitelisted in order to do so.

Sherlock

Fixed.

Issue M-6 `LeverEPTActionsbuyCollateralAndIncreaseLever()` will most certainly fail due to precision loss

Summary

Because of the precision loss in the conversion of borrowed amount (`addDebt`) to `deltaNormalDebt` with `wdiv(addDebt,rate)`, the newly generated internal credit for FIAT (`deltaDebt`) will most certainly always be 1 Wei less than `addDebt`.

As a result, the `buyCollateralAndIncreaseLever()` will fail due to insufficient credit in `codex.transferCredit()`.

Vulnerability Detail

When `buyCollateralAndIncreaseLever()` is called, it will take a flashloan from `flash` which will callback and call the internal function `addCollateralAndDebt()` with the amount of `addCollateral` and `borrowed` (as `addDebt`).

`codex.modifyCollateralAndDebt()` will then be called with `deltaNormalDebt=wdiv(addDebt,rate)`; In `codex.modifyCollateralAndDebt()`, the `deltaNormalDebt` will be convert back as `int256deltaDebt=wmul(v.rate,deltaNormalDebt);`.

The back and forth conversion will most certainly incur a precision loss, resulting in the `deltaDebt` to be 1 Wei smaller than `addDebt`.

Thus, the transaction will revert in `exitMoneta(creditor,addDebt);` as the `credit` will be insufficient (by 1 Wei).

Impact

`LeverEPTActionsbuyCollateralAndIncreaseLever()` will most certainly fail.

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L110-L136

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L140-L182

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverActions.sol#L143-L173

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/utils/Math.sol#L124-L128

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L297-L352

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/utils/Math.sol#L109-L113

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverActions.sol#L103-L110

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Moneta.sol#L58-L63

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L269-L278

Tool used

Manual Review

Recommendation

Consider using `wdivUp()` for `deltaNormalDebt=wdiv(addDebt,rate)`.

Team

Fix: https://github.com/fiatdao/actions/pull/12

Sherlock

Fixed.

Issue M-7 `Aer.lock` only locks itself.

Summary

`Aer.lock` doen't lock other `Aer`'s functionalities

Vulnerability Detail

`Aer.lock` sets `live` to 0, locks `surplusAuction` and `debtAuction` and clean the states.

```
function lock() external override checkCaller {
    if (live == 0) revert Aer__lock_notLive();
    live = 0;
    queuedDebt = 0;
    debtOnAuction = 0;
    surplusAuction.lock(codex.credit(address(surplusAuction)));
    debtAuction.lock();
    codex.settleUnbackedDebt(min(codex.credit(address(this)),
    ↪  codex.unbackedDebt(address(this))));
    emit Lock();
}
```

However, `Aer` only checks `live` in `Aer.lock`. The rest functions work as usual. `surplusAuction` and `debtOnAuction` can be reset.

Impact

`Aer` can still work after being locked. The only thing it cannot do after being locked is locking again.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Aer.sol#L212-L213

Tool used

Manual Review

Recommendation

Add live checks into other functions that should not work when `Aer` is locked.

Team

This is expected behavior. During shutdown (Tenebrae) Aer is used to track the global settlement process. https://github.com/fiatdao/fiat/blob/main/src/Tenebrae.sol#L341.

Sherlock

Acknowledged.

Issue M-8 Base interest can change while there is uncollected interest

Summary

`baseInterest` can be modified at any time, while there may be uncollected interest.

Vulnerability Detail

The `interestPerSecond` setter has a check to ensure that there is all interest has been collected prior to changing its value.

```
if (block.timestamp != vaults[vault].lastCollected) revert
↪  Publican__setParam_notCollected();
```

The `baseInterest` lacks the same check. Based on the comment definition, it is the "Global, ***per-second*** stability fee contribution", so it seems that it should only be modified when all interest has been collected too.

Impact

Base interest rate update is retroactively applied to uncollected interest, which may be undesirable.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L83 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L93-L97 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L46-L47

Tool used

Manual Review

Recommendation

Include the referenced conditional check in the setter for `baseInterest`.

Team

Interest collection happens every 3 days and every time a user interacts with the protocol. There could be a loss, but it would be insignificant. Additionally the current behavior solves the problem of the last depositor being unable to pay off accrued

interest because there's not enough FIAT in circulation. In a situation like this the DAO may set the rate to 0.

Sherlock

Acknowledged.

Issue M-9 Users can push any value to Collybus immediately if disputeWindow is less than proposeWindow.

Summary

Users can push any value to Collybus immediately if disputeWindow is less than proposeWindow.

Vulnerability Detail

If `disputeWindow` is less than `proposeWindow`, users can bypass conditions (`canShift`) called by shift function in OptimisticOracle.sol.

Suppose:

- current block.timestamp = 1000
- disputeWindow = 10
- proposeWindow = 100 (`disputeWindow` is less than `proposeWindow`)
- Nonce of current proposal is 0, or it's unable to dispute (`proposeWindow` exceeded)

Alice can push any value to Collybus immediately:

1. Because any users can craft nonce, Alice crafts a nonce that `decodeRoundTimestamp(nonce)=980` and calls `shift`.
2. `shift` function will check nonce by calling `canShift`, but the nonce crafted by Alice will pass canPropose(nonce) because `(block.timestamp-decodeRoundTimestamp(nonce)<=proposeWindow)` (`1000-980<=100`). Thus Alice updated the proposal with a malicious value.
3. Then Alice call `shift` again, now `canShift` will check that the nonce (it's `prevNonce` now) should be unable to dispute. But the nonce can pass the condition: `!canDispute(prevNonce): !(block.timestamp-decodeRoundTimestamp(nonce)<=disputeWindow)` (`!(1000-980<=10)`)

Finally, the proposal will be pushed to Collybus.

Impact

If disputeWindow is less than proposeWindow, attacker can push any malicious value to Collybus immediately in the same block. Users may get a wrong price when calling the `read` function in Collybus.sol.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/ChainlinkValidator.sol#L69

Tool used

Manual Review

Recommendation

Should check `disputeWindow>=proposeWindow` in constructor.

Team

Fixed in PR: https://github.com/fiatdao/delphi-v2/pull/8

Sherlock

Fixed. WIll revert if `proposeWindow>disputeWindow`.

Issue M-10 Unsafe casting to `uint88`

Summary

`fCashAmount` is unsafely casted to `uint88` in when buying and selling collateral in `VaultFCActions.sol`.

Vulnerability Detail

In both buying and selling collateral, the respective functions take `fCashAmount` as `uint256`. It is unsafely casted to `uint88` in the `_buyFCash()` and `_sellFCash()` functions. Even though `_sellfCash()` checks for the casting overflow, the casting has already performed, making it redundant.

The `deltaCollateral` calculated would be impacted as it retains the original value that could exceed `type(uint88).max`. As a POC, if we take fCashAmount to be `type(uint88.max)+1`, we would have vastly different values for `deltaCollateral` and `fCashAmount`:

- `deltaCollateral=309485009821345068724781056000000000`
- `fCashAmount=0`

Impact

The collateral to be credited / deducted can be made to be much greater / smaller than the `fCashAmount`. The likelihood of an exploit is low, because the collateral has to be transferred into and from the vault, which, in practice, would be too large an amount for most collateral tokens.

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultFCActions.sol#L342 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultFCActions.sol#L401 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultFCActions.sol#L504

Tool used

Manual Review

Recommendation

The `_buyFCash()` and `_sellFCash()` functions should take in `fCashAmount` as a `uint256` instead of `uint88`. The `_buyFCash()` should have the overflow check as well.

Team

Fix: https://github.com/fiatdao/actions/pull/13

Sherlock

Fixed.

Issue L-1 Have recovery function for failed token transfers in `_claimBond()`

Summary

Funds of failed `bondToken` transfers are unretrievable.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

The intention to have a try-catch for token transfers is to prevent blocking disputes. However, the downside is that funds will be permanently lost. For instance, it is possible that the `receiver` was input as the null address (blocked by OZ's ERC20 implementation), the token contract address which is blocked by some implementations, or a blacklisted address.

It would therefore be advisable to have some form of recovery method to sweep the funds out to the owner or a specified recipient.

Impact

Permanent lockup of funds from failed `bondToken` transfers.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L438-L444

Tool used

Manual Review

Recommendation

Introduce a fund recovery mechanism for failed token transfers.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue L-2 Incorrect `decodeNonce` natspec

Summary

The `decodeNonce()` natspec is incorrect.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Impact

Documentation correctness.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L322-L326 https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticChainlinkOracle.sol#L294-L298

Tool used

Manual Review

Recommendation

```
- /// @dev Reverts if the `disputeWindow` is still active
/// @param nonce Nonce of the previous proposal [keccak256(roundId,
↪  roundTimestamp), proposeTimestamp]
- @return dataHash [(roundId, roundTimestamp)]
+ @return dataHash keccak256 hash of `data`
- /// @return proposalTimestamp
+ /// @param proposalTimestamp Round timestamp [uint64]
```

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/24 as per Recommendation.

Sherlock

Fixed.

Issue L-3 Generalize data provider reference in `OptimisticOracle`

Summary

Avoid explicit mention of Chainlink in the general OptimisticOracle contract.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

It is recommended to avoid mentioning Chainlink explicitly in `OptimisticOracle` as subsequent optimistic oracle implementations may rely on other data providers for validation and disputes, resulting in different data formats and structures.

Impact

Incorrect assumption of `data` format and functionality from inline comments.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L168

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L174-L175

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L214

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L219

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L221

Tool used

Manual Review

Recommendation

```
- /// @notice Disputes a proposed value by fetching the correct value from the
 ↪  corresponding Chainlink feed.
+ /// @notice Disputes a proposed value by fetching the correct value from the
 ↪  implementation's data feed.
```

```
- /// @param nonce Nonce of the proposal being disputed [keccak256(roundId,
 ↪  roundTimestamp), proposeTimestamp]
+ /// @param nonce Nonce of the proposal being disputed [keccak256(data),
 ↪  proposeTimestamp]
- /// @param data Encoded chainlink data [roundId, roundTimestamp]
+ /// @param data Additional encoded data required for disputes

- /// @notice Validates `proposedValue` for given `nonce` via the corresponding
 ↪  Chainlink feed
+ /// @notice Validates `proposedValue` for given `nonce` via the
 ↪  implementation's data feed

- /// @param data Data containing additional Chainlink round data
 ↪  [(roundId[uint80], roundTimestamp[uint64])]
+ /// @param data Additional encoded data required for validation

- /// @return validValue Value that was retrieved from the Chainlink feed [wad]
+ /// @return validValue Value that was retrieved from the implementation's data
 ↪  feed [wad]
```

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/24 as per Recommendation.

Sherlock

Fixed.

Issue L-4 Long `disputeWindow` / large time difference between `shift()` calls may result in staleness of proposed value

Summary

A long `disputeWindow` value or significant time delay between shifts might result in staleness of the previous proposed value.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

Bonded proposers optimistically propose a value for the next spot price that can be disputed by other proposers within `disputeWindow` while pushing the previous spot price to Collybus.

Shifts have a minimum interval of `disputeWindow`. As such, a large `disputeWindow` value or significant time lapses between shifts mean that the previous spot price to be pushed might go stale.

Impact

Stale prices are pushed into Collybus.

Tool used

Manual Review

Recommendation

Ensure that `disputeWindow` is of a reasonable value and that proposers are sufficiently incentivised to keep `shift()` calls as close as possible.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue L-5 DoS `aer.startDebtAuction()` by `transferCredit(attacker,aer,1)`

Summary

The attacker can front run `aer.startDebtAuction()` and transfer 1 wei of credit to aer and make the `startDebtAuction()` transaction revert.

Vulnerability Detail

The attacker can front run `aer.startDebtAuction()` with a transfer of 1 wei credit to aer (`transferCredit(attacker,aer,1)`) to revert `startDebtAuction()` at Aer.sol#L149.

Because `startDebtAuction()` will revert with an error: "Aer__startDebtAuction_surplusNotZero" when `codex.credit(address(this))!=0`.

Impact

If `aer.startDebtAuction()` is initiated by a gov proposal or multi-sig transaction, the attacker would cause some operational hassle to recreate the proposal or get the multi-sig transaction signed again.

Code Snippet

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/Aer.sol#L141-L153

```
/// @notice Starts a debt auction
/// @dev Sender has to be allowed to call this method
/// Checks if enough debt exists to be put up for auction
/// debtAuctionBidSize > (unbackedDebt - queuedDebt - debtOnAuction)
/// @return auctionId Id of the debt auction
function startDebtAuction() external override checkCaller returns (uint256
↪  auctionId) {
    if (debtAuctionBidSize > sub(sub(codex.unbackedDebt(address(this)),
↪  queuedDebt), debtOnAuction))
        revert Aer__startDebtAuction_insufficientDebt();
    if (codex.credit(address(this)) != 0) revert
↪  Aer__startDebtAuction_surplusNotZero();
    debtOnAuction = add(debtOnAuction, debtAuctionBidSize);
    auctionId = debtAuction.startAuction(address(this), debtAuctionSellSize,
↪  debtAuctionBidSize);
    emit StartDebtAuction(debtOnAuction, auctionId);
}
```

Tool used

Manual Review

Recommendation

Consider creating a new contract that consumes all the credits first before `startAuction`:

```
function safeStartDebtAuction() external checkCaller returns (uint256 auctionId)
↪ {
    uint credit = codex.credit(aer);
    if (credit != 0) {
        aer.settleDebtWithSurplus(credit);
    }
    return aer.startDebtAuction();
}
```

This function can also be added as a new external method on the Aer contract:

```
function safeStartDebtAuction() external checkCaller returns (uint256 auctionId)
↪ {
    uint credit = codex.credit(address(this));
    if (credit != 0) {
        codex.settleUnbackedDebt(credit);
    }
    return startDebtAuction();
}
```

Team

Those methods are primarily executed via the DAO. The DAO is able to chain contract calls. In a proposal where it would auction off debt it would call settleDebtWithSurplus first. Acknowledged.

Sherlock

Acknowledged.

Issue L-6 Return value of `transferFrom()` isn't checked

Summary

The return value of `transferFrom()` isn't checked.

Vulnerability Detail

If it's only the FDT token that is to be exchanged for FIAT, then this issue is informational. However, if other tokens are to be accepted for the surplus FIAT auction (Eg. USDT), then it is recommended to check the return value of the `transferFrom()`, because some implementations return `false` instead of reverting.

Impact

Failed token transfers are considered to have been successful, effectively giving away FIAT tokens for free.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/SurplusAuction.sol#L154 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/SurplusAuction.sol#L157

Tool used

Manual Review

Recommendation

Use OpenZeppelin's SafeERC20 `safeTransferFrom()`, as using `require()` to check transfer return values could lead to issues with non-compliant ERC20 tokens which do not return a boolean value, like USDT.

```
token.safeTransferFrom(msg.sender, auctions[auctionId].recipient,
↪  auctions[auctionId].bid);
```

Team

Thanks! We know that this would only ever be FDT. Acknowledged.

Sherlock

Acknowledged.

Issue L-7 Lack of slippage control for the minimal amount of collateral bought in `takeCollateral()`

Summary

When the user calls `takeCollateral()` to buy from a CollateralAuction, the conditions may have changed during the time from the user pre-run and send the transaction til it getting minted, there is a chance that the auction is no longer favorable for the user.

Vulnerability Detail

There are existing checks to control the `maxPrice` and max collateral amount to buy (`collateralAmount`). But we believe the min collateral amount should also be controlled.

For example, if the user found that 10 ETH tokens were being sold at $1000 in an auction, they sent a transaction to `takeCollateral()`.

However, by the time the transaction gets minted, there is only 0.01 ETH left to be bought. The profit of this purchase will be less than the gas cost.

Checking if the `minCollateralToBuy` can be met and throwing a custom error if not, can prevent the user from sending such unfavorable transactions while estimating gas cost, or it can revert earlier to save gas if the transaction gets sent and minted.

Impact

The users may waste their gas to buy only a dust amount of collateral tokens.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/NoLossCollateralAuction.sol#L402-L432

```
function takeCollateral(
    uint256 auctionId, // Auction id
    uint256 collateralAmount, // Upper limit on amount of collateral to buy [wad]
    uint256 maxPrice, // Maximum acceptable price (Credit / collateral) [wad]
    address recipient, // Receiver of collateral and external call address
    bytes calldata data // Data to pass in external call; if length 0, no call is
 ↪  done
) external override checkReentrancy isStopped(3) {
    Auction memory auction = auctions[auctionId];

    if (auction.user == address(0)) revert
 ↪  NoLossCollateralAuction__takeCollateral_notRunningAuction();
```

```
    uint256 price;
    {
        bool done;
        (done, price) = status(auction);

        // Check that auction doesn't need reset
        if (done) revert NoLossCollateralAuction__takeCollateral_needsReset();
        // Ensure price is acceptable to buyer
        if (maxPrice < price) revert
↪   NoLossCollateralAuction__takeCollateral_tooExpensive();
    }

    uint256 collateralToSell = auction.collateralToSell;
    uint256 debt = auction.debt;
    uint256 owe;

    unchecked {
        {
            // Purchase as much as possible, up to collateralAmount
            // collateralSlice <= collateralToSell
            uint256 collateralSlice = min(collateralToSell, collateralAmount);
```

Tool used

Manual Review

Recommendation

Consider adding a new parameter `minCollateralAmount` and check if `auction.collateralToSell<minCollateralAmount`:

```
function takeCollateral(
    uint256 auctionId, // Auction id
    uint256 maxCollateralAmount, // Upper limit on amount of collateral to buy
↪   [wad]
    uint256 minCollateralAmount,
    uint256 maxPrice, // Maximum acceptable price (Credit / collateral) [wad]
    address recipient, // Receiver of collateral and external call address
    bytes calldata data // Data to pass in external call; if length 0, no call is
↪   done
) external override checkReentrancy isStopped(3) {
    Auction memory auction = auctions[auctionId];

    if (auction.user == address(0)) revert
↪   NoLossCollateralAuction__takeCollateral_notRunningAuction();

    uint256 collateralToSell = auction.collateralToSell;
```

```
+     if (collateralToSell < minCollateralAmount) revert
↪   NoLossCollateralAuction__takeCollateral_minCollateralAmount();

    uint256 price;
    {
        bool done;
        (done, price) = status(auction);

        // Check that auction doesn't need reset
        if (done) revert NoLossCollateralAuction__takeCollateral_needsReset();
        // Ensure price is acceptable to buyer
        if (maxPrice < price) revert
↪   NoLossCollateralAuction__takeCollateral_tooExpensive();
    }

    uint256 debt = auction.debt;
    uint256 owe;

    unchecked {
        {
            // Purchase as much as possible, up to collateralAmount
            // collateralSlice <= collateralToSell
            uint256 collateralSlice = min(collateralToSell, collateralAmount);
```

Team

The `auctionDebtFloor` protects against that (assuming it is set correctly). https://github.com/fiatdao/fiat/blob/main/src/auctions/CollateralAuction.sol#L396. It enforces a min. amount of outstanding debt to be covered.

Sherlock

Acknowledged.

Issue L-8 `Guarded._unsetRoot()` Wrong event emitted for `_unsetRoot()`

Summary

Based on the context, L85 should emit `BlockCaller` event instead of `AllowCaller` event.

Impact

The `emitAllowCaller(ANY_SIG,root);` in `_unsetRoot()` will be confused with the `AllowCaller(ANY_SIG,root)` events emitted in `_setRoot()`.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/utils/Guarded.sol#L49-L86

```
/// @notice Grant the right to call method `sig` to `who`
/// @dev Only the root user (granted `ANY_SIG`) is able to call this method
/// @param sig Method signature (4Byte)
/// @param who Address of who should be able to call `sig`
function allowCaller(bytes32 sig, address who) public override callerIsRoot {
    _canCall[sig][who] = true;
    emit AllowCaller(sig, who);
}

/// @notice Revoke the right to call method `sig` from `who`
/// @dev Only the root user (granted `ANY_SIG`) is able to call this method
/// @param sig Method signature (4Byte)
/// @param who Address of who should not be able to call `sig` anymore
function blockCaller(bytes32 sig, address who) public override callerIsRoot {
    _canCall[sig][who] = false;
    emit BlockCaller(sig, who);
}

/// @notice Returns if `who` can call `sig`
/// @param sig Method signature (4Byte)
/// @param who Address of who should be able to call `sig`
function canCall(bytes32 sig, address who) public view override returns (bool) {
    return (_canCall[sig][who] || _canCall[ANY_SIG][who] ||
↪   _canCall[sig][ANY_CALLER]);
}

/// @notice Sets the root user (granted `ANY_SIG`)
/// @param root Address of who should be set as root
```

```
function _setRoot(address root) internal {
    _canCall[ANY_SIG][root] = true;
    emit AllowCaller(ANY_SIG, root);
}

/// @notice Unsets the root user (granted `ANY_SIG`)
/// @param root Address of who should be unset as root
function _unsetRoot(address root) internal {
    _canCall[ANY_SIG][root] = false;
    emit AllowCaller(ANY_SIG, root);
}
```

Tool used

Manual Review

Recommendation

```
function _unsetRoot(address root) internal {
    _canCall[ANY_SIG][root] = false;
-     emit AllowCaller(ANY_SIG, root);
+     emit BlockCaller(ANY_SIG, root);
}
```

Team

Fix: https://github.com/fiatdao/fiat/pull/14

Sherlock

Fixed.

Issue L-9 `ChainlinkValidator` should has sanity checks for `proposeWindow` and `disputeWindow`

Summary

There is no check for `proposeWindow` and `disputeWindow` in `ChainlinkValidator`'s constructor.

Vulnerability Detail

`proposeWindow` and `disputeWindow` are set in `ChainlinkValidator`'s constructor. They are all immutable.

```
constructor(uint256 proposeWindow_, uint256 disputeWindow_) {
    proposeWindow = proposeWindow_;
    disputeWindow = disputeWindow_;
}
```

There are no sanity checks for `proposeWindow` and `disputeWindow`. They can be any value.

Impact

`proposeWindow` and `disputeWindow` are important in `ChainlinkValidator`. Bad values could cause problems.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/ChainlinkValidator.sol#L39-L42

Tool used

Manual Review

Recommendation

Add reasonable sanity checks, e.g. `disputeWindow` should greater than 1 day.

Team

Duplicate of https://github.com/SherlockAudit/fiat-dao/issues/31. Fixed in https://github.com/fiatdao/delphi-v2/pull/8

Sherlock

Not a duplicate. https://github.com/SherlockAudit/fiat-dao/issues/31 is about checking the relative values of `proposeWindow` and `disputeWindow` This issue is about checking their absolute values. Eg. ensuring that they are minimally above a certain value.

Team

We discussed adding absolute value checks for the windows but because of the low frequency of deploys and the fact that this will be done from a deployment script that will be tested as well we don't need to enforce a value interval for the two parameters. I updated this to acknowledged.

Sherlock

Acknowledged.

Issue L-10 `Publican.collect` should have a sanity check for `aer`

Summary

`aer` could be `address(0)` when calling `Publican.collect`. It leads to the loss of ccrued interest

Vulnerability Detail

`Publican.collect` calls `codex.modifyRate` to update the Vault's rate and collect accrued interest.

```
/// @notice Collects accrued interest from all Position on a Vault by updating
↪  the Vault's rate
/// @param vault Address of the Vault
/// @return rate Set rate
function collect(address vault) public override returns (uint256 rate) {
    if (block.timestamp < vaults[vault].lastCollected) revert
    ↪  Publican__collect_invalidBlockTimestamp();
    (, uint256 prev, , ) = codex.vaults(vault);
    rate = wmul(
        wpow(
            add(baseInterest, vaults[vault].interestPerSecond),
            sub(block.timestamp, vaults[vault].lastCollected),
            WAD
        ),
        prev
    );
    codex.modifyRate(vault, address(aer), diff(rate, prev));
    vaults[vault].lastCollected = block.timestamp;
    emit Collect(vault);
}
```

However, there is no check for `aer`. If `aer` is `address(0)`, the accrued interest will be lost.

Impact

The loss of accrued interest from all Position on vaults.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L142 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L478

Tool used

Manual Review

Recommendation

Add a sanity check in `Publican.collect`

```
    /// @notice Collects accrued interest from all Position on a Vault by
 ↪  updating the Vault's rate
    /// @param vault Address of the Vault
    /// @return rate Set rate
    function collect(address vault) public override returns (uint256 rate) {
        if (block.timestamp < vaults[vault].lastCollected) revert
 ↪  Publican__collect_invalidBlockTimestamp();
+       if (address(aer) == address(0)) revert Publican__collect_invalidAer();
        (, uint256 prev, , ) = codex.vaults(vault);
        rate = wmul(
            wpow(
                add(baseInterest, vaults[vault].interestPerSecond),
                sub(block.timestamp, vaults[vault].lastCollected),
                WAD
            ),
            prev
        );
        codex.modifyRate(vault, address(aer), diff(rate, prev));
        vaults[vault].lastCollected = block.timestamp;
        emit Collect(vault);
    }
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue L-11 Incorrect overflow checks

Summary

`fyTokenAmount` and `underlierAmount` are exclusive of `type(uint128.max)` when it should be inclusive.

Vulnerability Detail

`fyTokenAmount` and `underlierAmount` are of type `uint256`. There are overflow checks to prevent them from exceeding `type(uint128).max)`.

```
if (underlierAmount >= type(uint128).max) revert
↪  VaultFYActions__buyCollateralAndModifyDebt_overflow();
if (fyTokenAmount >= type(uint128).max) revert
↪  VaultFYActions__sellCollateralAndModifyDebt_overflow();
```

However, it excludes `type(uint128).max)` when in fact it should be allowed, as per the fyToken implementation.

```
/// @dev Safely cast an uint256 to an uint128
function u128(uint256 x) internal pure returns (uint128 y) {
   require (x <= type(uint128).max, "Cast overflow");
   y = uint128(x);
}
```

Impact

Reduced value ranges for `fyTokenAmount` and `underlierAmount`.

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultFYActions.sol#L91-L92 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultFYActions.sol#L133-L134

Tool used

Manual Review

Recommendation

Because Yield Protocol uses `uint128`, it would be better to change the types of related variables `SwapParams.minAssetOut`, `underlierAmount` and `fyTokenAmount` from `u`

`int256` to `uint128` so that overflow checks can be avoided altogether.

Team

`type(uint128).max)` will trigger the Arithmetic over/underflow error within the Yield Protocol contracts. It has to be excluded. I might consider a refactor to reduce LOC of code. But I don't see the need for a fix.

Sherlock

Acknowledged.

Issue L-12 Duplicate approval to balancer vault for swaps

Summary

There is a duplicate token approval in `_sellPToken()` given to the balancer vault.

Vulnerability Detail

Approval is given once at the beginning of the `_sellPToken()` function in the `VaultEPTActions.sol` contract.

```
// approve Balancer to transfer PToken
IERC20(swapParams.assetIn).approve(swapParams.balancerVault, pTokenAmount);
```

Some lines later, another approval is made if `swapParams.approve` is non-zero.

```
if (swapParams.approve != 0) {
  // approve balancer vault to transfer pTokens on behalf of proxy
  IERC20(swapParams.assetIn).approve(swapParams.balancerVault,
↪   swapParams.approve);
}
```

Impact

Some tokens (Eg. USDT) will revert from approving non-zero to non-zero allowances. Hence, the parent function `sellCollateralAndModifyDebt()` will revert because of the double approval.

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultEPTActions.sol#L220-L221 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultEPTActions.sol#L238-L241

Tool used

Manual Review

Recommendation

The approvals should be merged into one to save gas so that approval is given just once. It is also recommended to use the SafeERC20. Note that OpenZeppelin has stated that the `safeApprove()` function is deprecated because it has issues similar to the ones found in IERC20.approve(), and its usage is discouraged.

Whenever possible, use safeIncreaseAllowance() and safeDecreaseAllowance() instead.

Team

Fix: https://github.com/fiatdao/actions/pull/14

Sherlock

Fixed. Approval with `pTokenAmount` has been removed.

Issue L-13 Incorrect referenced commit for `StateProofHandler.sol`

Summary

The `StateProofHandler.sol` contract points to the referenced library with a specific commit, but has differing implementations.

Vulnerability Detail

The `StateProofHandler.sol` contract points to a specific commit to the referenced library which can be found in LidoFinance's repo. Unfortunately, the referenced commit has the `verifyStateProof()` function removed in an earlier commit. A later commit to the contract also contained a change: adding an explicit require statement for easier debugging that might be worth including.

Impact

Code clarity.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/StateProofVerifier.sol

Tool used

Manual Review

Recommendation

Either make a loose reference to the original library (ie. without specifying the commit), or reference the earlier commit prior to the `verifyStateProof()` function removal. Consider including the explicit require check for easier debugging and make reference to that commit as well.

Team

Fixed in PR:https://github.com/fiatdao/delphi-v2/pull/9

Sherlock

Fixed. The require check has been added and `verifyStateProof()` function removed.

Issue L-14 Lack of validation check for `rateType` parameter

Summary

`rateType` lacks sufficient validation checks when set.

Vulnerability Detail

`rateType` is allowed to take on the range of `uint96` in `setRateConfig()`, but is subsequently casted to enum `RateType` which has only 2 values. Hence, if `rateType>1`, `_push()` will revert in the following line:

```
_push(RateType(rateConfig.rateType), rateId, prevValue);
```

Impact

Prices cannot be pushed until the rate configuration has been unset and set again.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L156

Tool used

Manual Review

Recommendation

Validate `rateType` to be only `0` or `1` in `setRateConfig()`.

```
if (rateType > 1) revert OptimisticOracle__setRateConfig_invalidRateType();
```

Team

Fixed in PR: https://github.com/fiatdao/delphi-v2/pull/10

Sherlock

Fixed.

Issue I-1 Inconsistent variable name `proposeTimestamp` and `proposalTimestamp`

Summary

A couple of variable names are used to refer to the proposal timestamp.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

Most of the codebase refers to the proposal timestamp as `proposeTimestamp` (eg. `_encodeProposeTimestamp()` / `_decodeProposeTimestamp()`), but `decodeNonce()` has the variable name `proposalTimestamp` instead.

Impact

Variable naming consistency.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L322

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L324

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L326

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticOracle.sol#L331

Tool used

Manual Review

Recommendation

For consistency, change `proposalTimestamp` to `proposeTimestamp`.

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/24 as recommended.

Sherlock

Fixed.

Issue I-2 Perform dispute window check before nonce calculation in `encodeNonce()`

Summary

Reduce the gas consumption of failure cases by performing the revert check prior to nonce calculation.

Note: This finding was part of the delphi-v2 refactor review which took place from Oct 3rd to Oct 5th.

Vulnerability Detail

In `encodeNonce()`, the check if the previous proposal is within disputeWindow should be performed prior to the nonce calculation. This helps to save gas in failure cases.

Impact

Higher gas usage.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/181d85bd82ea1e5d468767e5368111734c39b908/src/OptimisticChainlinkOracle.sol#L272-L292

Tool used

Manual Review

Recommendation

```
function encodeNonce(bytes32 prevNonce, bytes memory data)
    public
    view
    override(OptimisticOracle)
    returns (bytes32 nonce)
{
    // Revert if prev. proposal is still within `disputeWindow`
    if (
        prevNonce != 0 &&
        sub(block.timestamp, _decodeProposeTimestamp(prevNonce)) <=
        disputeWindow
    ) {
        revert OptimisticChainlinkOracle__encodeNonce_activeDisputeWindow();
    }
```

```
    // Pack the propose timestamp in with the data hash
    nonce = _encodeProposeTimestamp(
        keccak256(data),
        uint64(block.timestamp)
    );
}
```

Team

Partially fixed in https://github.com/fiatdao/delphi-v2/pull/24. This is not a full fix because encodeNonce() was refactored a bit and the code is not the same.

Sherlock

Acknowledged partial fix due to refactor.

Issue I-3 Leftover FDT on the `DebtAuction` contract cannot be retrieved

Summary

A certain amount of unused FDT tokens will remain in the DebtAuction contract after `closeAuction()`.

Vulnerability Detail

In `closeAuction()`, the FDT tokens will be transferred to the highest bidder.

However, given the natural of the `DebtAuction`, the actual amount needed in `closeAuction()` will be fewer than the initial amount transferred in to the `DebtAuction` to start the auction.

Therefore, a certain amount of FDT tokens will remain in the `DebtAuction`.

Impact

Leftover FDT will be frozen in the contract until the next auction, and can not be returned to circulation by other means before then.

There will be extra hassles if the DebtAuction contract will ever be retired. An auction must be done to retrieve the FDT tokens, or they will be frozen in the contract forever.

Code Snippet

https://github.com/fiatdao/fiat/blob/b47e4a3c4800f7dd89a0531b6acf3bbe729e9860/src/auctions/DebtAuction.sol#L184-L193

```
function closeAuction(uint256 auctionId) external override {
    if (live == 0) revert DebtAuction__closeAuction_notLive();
    if (
        !(auctions[auctionId].bidExpiry != 0 &&
            (auctions[auctionId].bidExpiry < block.timestamp ||
                auctions[auctionId].auctionExpiry < block.timestamp))
    ) revert DebtAuction__closeAuction_notFinished();
    token.transfer(auctions[auctionId].recipient,
↪   auctions[auctionId].tokensToSell);
    delete auctions[auctionId];
}
```

Tool used

Manual Review

Recommendation

Consider sending all the remaining FDT balance back to the governance address.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-4 Potential precision loss when converting `tokenAmount` to WAD scale in `Vault.solexit()`

Summary

`exit()` will normalize the input `amount` (in `tokenScale`) to `18` (`wad`).

As a result, when `tokenScale>1e18`, the amount of balance taken from the user (`wad`) can be fewer than expected due to precision loss.

Vulnerability Detail

There are certain tokens with `decimals>18`, e.g. `NEAR` with decimals == 24.

PoC

Given:

- token.decimals: 24 $\implies$ tokenScale == 1e24

When:

- `exit(tokenId,msg.sender,999999)`
  - $wat = \frac{amount \cdot WAD}{tokenScale} = \frac{999999 \cdot 10^{18}}{10^{24}} = 0$

Then:

- `msg.sender`' balance on `codex` will remain unchanged: `codex.modifyBalance(address(this),0,msg.sender,-int256(0));`
- `msg.sender` will receive `999999` token: `IERC20(token).safeTransfer(user,999999);`

Impact

Given that the gas cost for such an attack is high, and the profit is only `1/1e18*tokenScale` wei of the token, which should be negligible unless the token is super expensive.

Code Snippet

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/Vault.sol#L120-L129

```
function exit(
    uint256, /* tokenId */
    address user,
    uint256 amount
) external virtual override {
```

```
    int256 wad = toInt256(wdiv(amount, tokenScale));
    codex.modifyBalance(address(this), 0, msg.sender, -int256(wad));
    IERC20(token).safeTransfer(user, amount);
    emit Exit(user, amount);
}
```

https://github.com/fiatdao/fiat/blob/b8406c29638b9f6e598ad6d961583df5b0a719a5/src/utils/Math.sol#L124-L128

```
function wdiv(uint256 x, uint256 y) pure returns (uint256 z) {
    unchecked {
        z = mul(x, WAD) / y;
    }
}
```

Tool used

Manual Review

Recommendation

Consider using rounding up rather than rounding down.

Team

Thanks! We did not onboard any tokens yet with a precision greater than 1e18. Though when we do we'll definitely consider that. Acknowledged.

Sherlock

Acknowledged.

Issue I-5 Redundant `wmul` operation with `WAD`

Summary

`wmul(x,WAD)` is equivalent to `x`, so the former expression should be replaced with the latter to avoid performing a redundant multiplication and division operation.

Vulnerability Detail

```
wmul(x, WAD)
= x * WAD / WAD
= x
```

Hence, `wmul(debt,WAD)` should be simplified to `debt`.

Impact

Higher gas costs.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Tenebrae.sol#L221

Tool used

Manual Review

Recommendation

```
- return wdiv(sub(collateral, lostCollateral[vault][tokenId]), wmul(debt, WAD));
+ return wdiv(sub(collateral, lostCollateral[vault][tokenId]), debt, WAD);
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-6 Use "" instead of `newbytes(0)`

Summary

It's cheaper to use "" instead of `newbytes(0)`.

Vulnerability Detail

Using "" is more gas efficient that `newbytes(0)` for the `bytesdata` field.

Impact

Higher gas costs.

Code Snippet

https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/Vault.sol#L258 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/Vault.sol#L273 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/Vault1155Actions.sol#L56

Tool used

Manual Review

Recommendation

```
- IERC1155(token).safeTransferFrom(from, address(this), tokenId, amount, new
↪  bytes(0));
+ IERC1155(token).safeTransferFrom(from, address(this), tokenId, amount, "");

- IERC1155(token).safeTransferFrom(msg.sender, address(this), tokenId, amount,
↪  new bytes(0));
+ IERC1155(token).safeTransferFrom(msg.sender, address(this), tokenId, amount,
↪  "");

- IERC1155(token).safeTransferFrom(from, address(this), tokenId, amount, new
↪  bytes(0));
+ IERC1155(token).safeTransferFrom(from, address(this), tokenId, amount, "");
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-7 `if(deltaNormalDebt<0)` branch can be nested in `if(deltaNormalDebt!=0)` branch

Summary

The `if(deltaNormalDebt<0)` branch can be nested into the `if(deltaNormalDebt!=0)` branch as a gas optimization.

Vulnerability Detail

Since `deltaNormalDebt<0` is a subset of `deltaNormalDebt!=0`, its branch can be moved inside. This optimizes the case where deltaNormalDebt is 0.

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultActions.sol#L126-L130

Tool used

Manual Review

Recommendation

```
if (deltaNormalDebt != 0) {
   publican.collect(vault);

  if (deltaNormalDebt < 0) {
      // add due interest from normal debt
      (, uint256 rate, , ) = codex.vaults(vault);
      enterMoneta(creditor, uint256(-wmul(rate, deltaNormalDebt)));
  }
}
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-8 Anti-pattern design of access control

Summary

The design of access control with `Guarded.sol` is anti-pattern.

Vulnerability Detail

While the system has a multi-level access control system with `Guarded.sol` and the `checkCaller` modifier, it's hard to tell the different roles and privileges from the source code of the smart contracts.

Because the actual privileges (who can call which function) are set after the deployment in a very flexible way:

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/utils/Guarded.sol#L53-L56

```
function allowCaller(bytes32 sig, address who) public override callerIsRoot {
    _canCall[sig][who] = true;
    emit AllowCaller(sig, who);
}
```

We believe this design is anti-pattern as OpenZeppelin's `AccessControl` library is already widely adopted.

It provides visibility of different roles and their privileges, among other best practices:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol

Impact

Visibility of different roles and their privileges is diminished.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/utils/Guarded.sol#L43-L47

```
modifier checkCaller() {
    if (canCall(msg.sig, msg.sender)) {
        _;
    } else revert Guarded__notGranted();
}
```

Tool used

Manual Review

Recommendation

Consider adopting OZ's `AccessControl` or improve the visibility of different roles and privileges in `Guarded.sol`.

Team

We considered using the AccessControl lib from OZ though we couldn't find a use case where we would have well defined persistent roles. E.g. In the long run we have the DAO as root and different smaller contracts with a unique set of privileges to call certain methods on other contracts. Roles make sense if you have a relationship were multiple different addresses / accounts should have the same set of permissions. The only instance where this would be the case in our protocol would be for the Vault contracts. The goal is to have access control as flexible, efficient and minimal as possible. The AccessControl lib does not meet those goals for us.

Sherlock

Acknowledged.

Issue I-9 Consider introducing an alternative `ConvexDecrease` formula for the Dutch Auction price

Summary

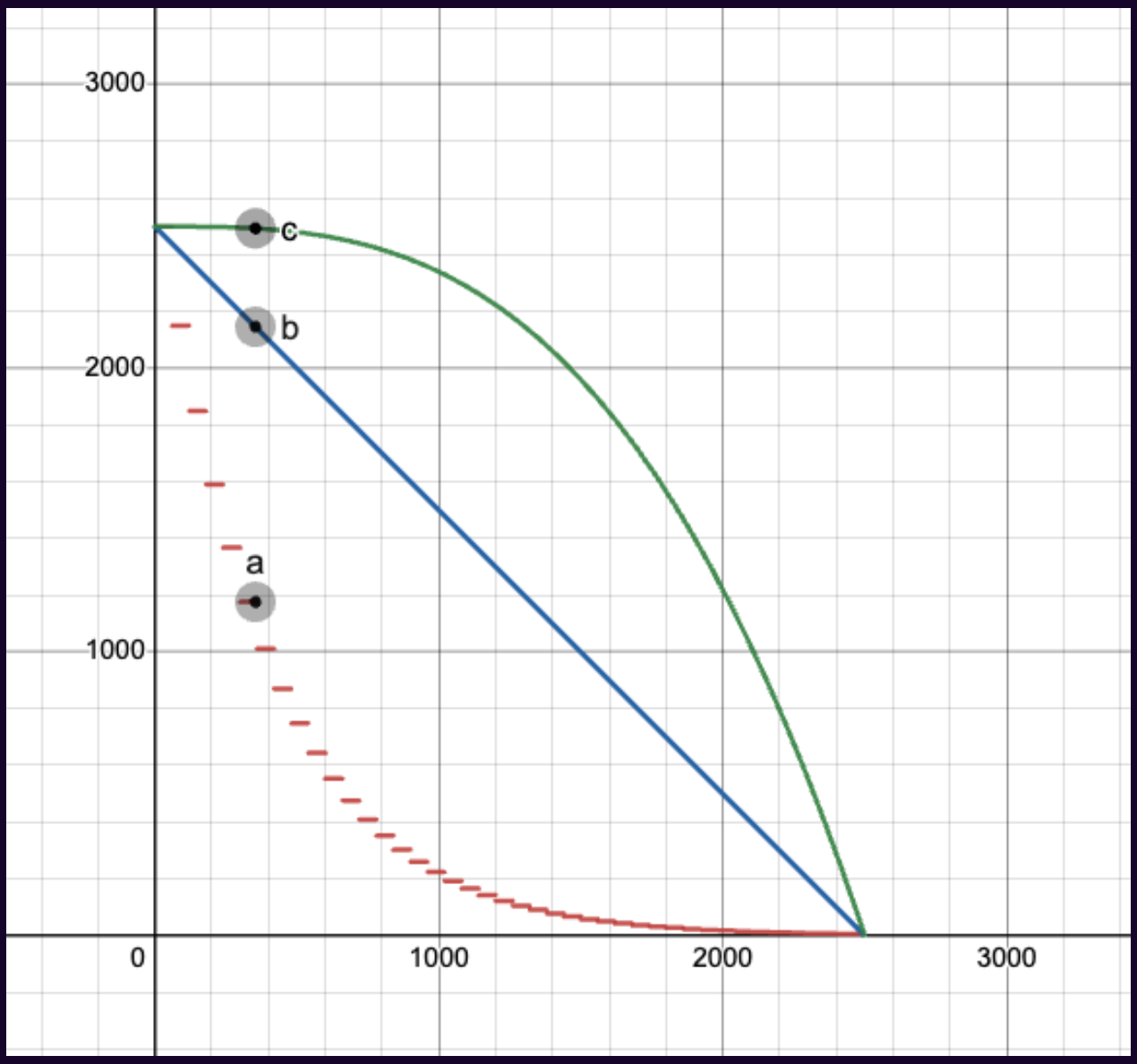
The Stairstep Exponential Decrease formula is inefficient in comparison with LinearDecrease and another alternative formula:`ConvexDecrease`.

Vulnerability Detail

The current implementation of `PriceCalculator.sol` provides 3 kinds of formulas: LinearDecrease, "Stairstep Exponential Decrease", and ExponentialDecrease for the price of different vaults.

Among the 3 formulas, we believe the "Stairstep Exponential Decrease" formula is inefficient in comparison with the **LinearDecrease** and another alternative formula (`ConvexDecrease`):

1. **LinearDecrease (Blue)**: Classic liner decrease;
2. **ConvexDecrease (Green)**: The price drops slowly at first, and accelerates over time.



https://www.desmos.com/calculator/bwaukhdqm3

As we demonstrated in the chart above, if the start price of the auction is $2500 and the fair market price is $2000;

- With the Stairstep formula (Red)
  - The price will skip $2000 and drop from $2150 suddenly to below 2000 at 1894 by time 120;
  - The auction will be sold $106 below the fair market price;
- With the Liner formula (Blue)
  - The price will decrease linearly to near $2000 by time 500
  - The auction will be sold at about the fair market price but at later time;

- With the Convex formula (Green)
  - The price will decrease slowly to near `$2000` by time `1460`.
  - There will be a longer time period for the auction price to stay in around the fair market price
  - The auction will be sold at an even later time.

Convex formula can better than Concave and Liner formulas Considering the fact that we are using the oracle price (`IVault.fairPrice()`) multiply a configured `multiplier` as the start price for the auction:

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/CollateralAuction.sol#L266-L288

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/CollateralAuction.sol#L266-L310

> multiplicative factor to increase the start price, and `redemptionPrice` is a reference per Credit.

When the `multiplier` is relatively small, or the oracle price is already lower than the fair market price, it's very likely the start price is already near the market price.

Therefore, the ConvexDecrease formula is more suitable as it will slowly adjust the price in the beginning, and only to decrease the price quickly when there are no takers at the earlier prices.

In other words, the ConvexDecrease formula will sell the assets at a higher price if the market price is near the oracle price, which means that the oracle is working properly now.

And if the oracle is not working properly and the market price is much lower, the ConvexDecrease formula will start to adjust the price swiftly to sell the assets as soon as possible.

Smooth formula is better than step formula We also believe making the price drop smoothly over time (`ExponentialDecrease`) is better than by step (`StairstepExponentialDecrease`), as the liquidation auctions will most likely be bought by bots.

Step prices only make sense for humans, it does not make sense for bots as they can just calculate the price in real-time.

When the price drops by step, there must be a sufficient price drop by each step, otherwise the price can not be changed quickly enough, which means the auction will last for too long if the price drop of each step is too small.

This makes it harder to have the assets sold close to the current market price.

A smooth price formula does not have such problems, therefore it allows the assets to be sold at the best price possible.

Impact

Using `StairstepExponentialDecrease` for the auction may result in the assets being sold at a suboptimal price.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/CollateralAuction.sol#L562-L571

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/NoLossCollateralAuction.sol#L556-L565

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/PriceCalculator.sol#L47-L57

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/PriceCalculator.sol#L103-L113

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/PriceCalculator.sol#L161-L170

Tool used

Manual Review

Recommendation

Consider introducing an alternative ConvexDecrease formula for the Dutch Auction price.

The ConvexDecrease formula can be used for assets with deep liquidity and relatively low volatility. The multiplier of the start price can then be set lower and still make sure the auction can be sold at a fair price rather quickly.

Team

Thanks! We are taking a look at the gradual dutch auction design as well. This contract will likely get revamped in the future!

Sherlock

Acknowledged.

Issue I-10 Missing sanity check for `vault` and `tokenId` in `NoLossCollateralAuctionActionstakeCollateral()`

Summary

Vulnerability Detail

If the `auctionId` doesn't match `vault` and `tokenId`, the transaction can still go through, but the collateral bought will not be sent to the `from` address as expected, because the `bought` amount will be `0`.

Such sanity check will make the `NoLossCollateralAuctionActionstakeCollateral()` function less error-prone.

Impact

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/auction/NoLossCollateralAuctionActions.sol#L68-L104

Tool used

Manual Review

Recommendation

Consider adding a check to ensure the `vault` and `tokenId` params match the `auctionId`:

```
    // transfer bought collateral to recipient
    uint256 bought = wmul(sub(codex.balances(vault, tokenId, address(this)),
↪   balance), IVault(vault).tokenScale());
+    if (bought > 0) revert
↪   NoLossCollateralAuction__takeCollateral_nonZeroBought();
    IVault(vault).exit(tokenId, recipient, bought);
}
```

Team

In this case the user would have to send another transaction - to withdraw the bought collateral with the right vault and tokenId. We will probably deprecate this contract soon though

Sherlock

Acknowledged.

Issue I-11 Lack of incentive for proposing new values with calling `shift()`

Summary

There is no enough incentive for the proposers to update the price regularly.

Vulnerability Detail

> Proposers are not rewarded for proposing new values and instead are only compensated in the event that they call the dispute function, as dispute is a gas intensive operation due to its computation of the expected value on-chain. Compensation is sourced from the bond put up by the malicious proposer.

The system relies on the Proposers to call `shift()` to keep the price up-to-date.

However, there is no incentive for a proposer to call `shift()` and update the price. On the contrary, given the fact that calling `shift()` still has some gas cost and the potential risk of being disputed and losing the bond, the proposers are disincentivized to `shift()` the price.

Not to mention that they also need to bond with a certain amount of funds staked before becoming a Proposer, which further disincentives the updates of the price feeds.

On the other hand, the liquidators, do have some motivation to become proposer and update the price, but that's only the case when the price goes downwards and opens an opportunity for liquidations.

Impact

We believe the lack of incentive to update the price can undermine the design of the Delphi v2 system, resulting in the price feeds not being updated in time as expected.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/main/src/OptimisticOracle.sol#L187-L237

Tool used

Manual Review

Recommendation

Consider introducing an off-chain retrospective rewarding system, in which the Proposers can get rewarded without adding any gas cost to the `shift()` call.

Team

For this version this was a conscious decision to leave it out. Incentives are coordinated off-chain. This reduces the on-chain gas footprint and makes the billing much more precise because we can see the exact gas cost that a given proposer paid.

Sherlock

Acknowledged.

Issue I-12 Use of unreleased version of OpenZeppelin contracts

Summary

The current codebase is using an unreleased version of OpenZeppelin contracts (the master branch), which is not preferred.

Vulnerability Detail

Impact

Unreleased version may include unaudited codes or breaking changes, which can potential introduce security vulnerabilities.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/.git/modules/lib/openzeppelin-contracts/HEAD#L1

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/9bded169e8f765f2e9279dca599c2175ce81dbbd/CHANGELOG.md?plain=1#L3

```
# Changelog

## Unreleased
```

Tool used

Manual Review

Recommendation

Consider specify a release such as `v4.7.3`, instead of using the `master` branch.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-13 Lack of two-step procedure for `Codex.sollock()`

Summary

`Codex.sollock()` can be done in one transaction and it's irreversible which is error-prone.

Vulnerability Detail

`Codex` is an essential component of the system and once `Codex.lock()` is called, it will immediately and irreversible disable some of its core functions, including:

- `setParam()`
- `modifyCollateralAndDebt()`
- `modifyRate()`

Such important operation should not be done in one-step and effective immediately.

Impact

If `Codex.lock()` is called mistakenly, the `Codex` contract will permanently will lose some of its core functions.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L499-L504

```
/// @notice Locks the contract
/// @dev Sender has to be allowed to call this method
function lock() external override checkCaller {
    live = 0;
    emit Lock();
}
```

Tool used

Manual Review

Recommendation

Consider using a timelock, and/or creating two roles: one to propose the shutdown, another to reject or approve it.

Team

All setParam methods can be called either by DAO which enforces a time delay of around a week or the Guard contracts and the multisig. Which will be phased out after the Gov upgrade.

Sherlock

Acknowledged.

Issue I-14 Consider making `flash` immutable in `FlashLoanReceiverBase`

Summary

Making `flash` immutable in `FlashLoanReceiverBase` can save gas and make it compatible with upgradable contracts.

Vulnerability Detail

The current implementation of `FlashLoanReceiverBase` is not compatible with upgradable contracts as it includes a storage variable `flash`.

In other words, an upgradable contract can not inherit `FlashLoanReceiverBase`.

Since `flash` will not be updated, it can be changed to `immutable` to save gas, and making the abstract contract `FlashLoanReceiverBase` compatible with upgradable contracts.

Impact

Higher gas cost and incompatible with upgradable contracts.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Flash.sol#L164-L183

```
abstract contract FlashLoanReceiverBase is ICreditFlashBorrower,
↪   IERC3156FlashBorrower {
    Flash public flash;

    bytes32 public constant CALLBACK_SUCCESS =
↪   keccak256("ERC3156FlashBorrower.onFlashLoan");
    bytes32 public constant CALLBACK_SUCCESS_CREDIT =
↪   keccak256("CreditFlashBorrower.onCreditFlashLoan");

    constructor(address flash_) {
        flash = Flash(flash_);
    }

    function approvePayback(uint256 amount) internal {
        // Lender takes back the FIAT as per ERC3156 spec
        flash.fiat().approve(address(flash), amount);
    }

    function payBackCredit(uint256 amount) internal {
```

```
        // Lender takes back the FIAT as per ERC3156 spec
        flash.codex().transferCredit(address(this), address(flash), amount);
    }
}
```

Tool used

Manual Review

Recommendation

Change to: `Flashpublicimmutableflash;`

Team

Fix: https://github.com/fiatdao/fiat/pull/12

Sherlock

Fixed.

Issue I-15 `VaultFYmaturity()` Cache fyToken's `maturity` in storage can save gas

Summary

FyToken's `maturity` can be saved as a storage variable to save gas.

Vulnerability Detail

FYToken's `maturity` is `immutable`, therefore it can be saved in storage to save gas from external calls.

https://github.com/yieldprotocol/vault-v2/blob/master/packages/foundry/contracts/FYToken.sol#L40

```
uint256 public immutable override maturity;
```

The current implementation involves 1 SLOAD (`token`) and 1 external call (`IFYToken.maturity()`), the improved implementation has only 1 SLOAD.

Impact

Higher gas cost.

Code Snippet

https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultFY.sol#L150-L155

```
function maturity(
    uint256 /* tokenId */
) public view virtual override returns (uint256) {
    // avoid sLOAD
    return IFYToken(token).maturity();
}
```

Tool used

Manual Review

Recommendation

Consider saving the `maturity` of the FYToken in `initialize()` and return it in `maturity()`.

Team

Fix: https://github.com/fiatdao/vaults/pull/7

Sherlock

Fixed.

Issue I-16 Misleading comments

Summary

There are a few wrong/misleading comments across the codebase.

Vulnerability Detail

1. `VaultFY.solfairPrice()` has no requirements for the caller, while the comment indicates that:

> Caller has to set allowance for this contract

2. `Flash.solflashFee()` reverts with a custom error when the token is not `FIAT`, while the comment indicates that:

> If `token` is not FIAT then 0 is returned

3. `Tenebrae.soloffsetPosition()` and `NoLossCollateralAuction.sol_getPrice()` fetches the current fair value, while the comment indicates that :

> // get price at maturity

Impact

Code Snippet

https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultFY.sol#L159-L171 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Flash.sol#L92-L103 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Tenebrae.sol#L278-L279 https://github.com/SherlockAudit/fiat-dao/blob/3e22c310cad47ab30d4adf8e2e1f582c8f9474c1/NoLossCollateralAuction.sol#L256-L259

Tool used

Manual Review

Recommendation

Update/remove the comments.

Team

Fix:

- https://github.com/fiatdao/vaults/pull/8
- https://github.com/fiatdao/fiat/pull/13

Sherlock

Fixed.

Issue I-17 `RLPReader.solcopy()` Return if `len=0` before L394 can save gas

Summary

The `RLPReader.solcopy()` function can return earlier to save gas.

Vulnerability Detail

When `len==0` after the for loop at L383-392, there is no need to continue and copy the left over bytes.

Therefore, return earlier can save gas.

Impact

Higher gas cost.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/9f9d32e05cd4b49b643487c19ffb26d5856af34b/src/validators/utils/RLPReader.sol#L375-L405

```
function copy(
    uint256 src,
    uint256 dest,
    uint256 len
) private pure {
    if (len == 0) return;

    // copy as many word sizes as possible
    for (; len >= WORD_SIZE; len -= WORD_SIZE) {
        assembly {
            mstore(dest, mload(src))
        }

        unchecked {
            src += WORD_SIZE;
            dest += WORD_SIZE;
        }
    }

    // left over bytes. Mask is used to remove unwanted bytes from the word
    uint256 mask;
    unchecked {
        mask = 256**(WORD_SIZE - len) - 1;
```

```
    }

    assembly {
        let srcpart := and(mload(src), not(mask)) // zero out src
        let destpart := and(mload(dest), mask) // retrieve the bytes
        mstore(dest, or(destpart, srcpart))
    }
}
```

Tool used

Manual Review

Recommendation

Consider adding `if(len==0)return;` before L394.

Team

I tested this change on the same block with the same test transaction and without the change, the total gas usage is 3983253, with the change it is 3983219(-34). Because the improvement is small we prefer keeping the contract as close to the original as possible.

Sherlock

Acknowledged.

Issue I-18 Unused imports

Summary

Some contracts have imports that are unused.

Vulnerability Detail

The following imports are unused by the contracts:

- VaultActions.sol

```
import {IERC1155} from "openzeppelin/contracts/token/ERC1155/IERC1155.sol";
import {ERC1155Holder} from
↪  "openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol";
```

- VaultEPTActions.sol

```
import {ICodex} from "fiat/interfaces/ICodex.sol";
import {IMoneta} from "fiat/interfaces/IMoneta.sol";
import {IFIAT} from "fiat/interfaces/IFIAT.sol";
```

- Lever20Actions.sol

```
import {ICodex} from "fiat/interfaces/ICodex.sol";
import {IMoneta} from "fiat/interfaces/IMoneta.sol";
import {IFIAT} from "fiat/interfaces/IFIAT.sol";
import {WAD, toInt256, mul, div, wmul, wdiv} from "fiat/utils/Math.sol";
```

- LeverActions.sol

```
import {SafeERC20} from
↪  "openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

- LeverEPTActions.sol

```
import {ICodex} from "fiat/interfaces/ICodex.sol";
import {IMoneta} from "fiat/interfaces/IMoneta.sol";
import {IFIAT} from "fiat/interfaces/IFIAT.sol";
import {IPublican} from "fiat/interfaces/IPublican.sol";
```

- VaultFCActions.sol

```
import {ICodex} from "fiat/interfaces/ICodex.sol";
import {IMoneta} from "fiat/interfaces/IMoneta.sol";
import {IFIAT} from "fiat/interfaces/IFIAT.sol";
```

- VaultEPT.sol

```
import {Clones} from "openzeppelin/contracts/proxy/Clones.sol";
import {VaultFactory} from "./VaultFactory.sol";
```

- VaultFC.sol

```
import {Vault1155} from "./Vault.sol";
```

- VaultFY.sol

```
import {Clones} from "openzeppelin/contracts/proxy/Clones.sol";
import {VaultFactory} from "./VaultFactory.sol";
```

- Moneta.sol

```
import {WAD, wmul} from "./utils/Math.sol";
```

- Collybus.sol

```
import {ICodex} from "./interfaces/ICodex.sol";
```

- Flash.sol

```
import {WAD, add, sub, wmul} from "./utils/Math.sol";
```

Impact

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultActions.sol#L5-L6 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultEPTActions.sol#L7 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultEPTActions.sol#L9-L10 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/Lever20Actions.sol#L7 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/Lever20Actions.sol#L9-L11 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverActions.sol#L5 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L7 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L9-L10 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L12 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultFCActions.sol#L8-L10 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a

61f505722/src/VaultEPT.sol#L8 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultEPT.sol#L16 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultFC.sol#L16 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultFY.sol#L8 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultFY.sol#L16 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Moneta.sol#L10 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Collybus.sol#L6 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Flash.sol#L11

Tool used

Manual Review

Recommendation

Remove all unused imports.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-19 Cache array length

Summary

`_slotHashes.length` is called multiple times. It would be cheaper to cache it in a variable.

Vulnerability Detail

It would be more gas efficient to cache the value of `_slotHashes.length` instead of retrieving it multiple times.

Impact

Higher gas costs

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/StateProofVerifier.sol#L73-L83

Tool used

Manual Review

Recommendation

```
uint256 slotHashesLength = _slotHashes.length;
```

Team

I have tested this change and it seems like the optimization is done either way by the compiler, caching the array length leads to no delta in gas consumption so I'd rather keep the delta to the original contract as small as possible.

Sherlock

Acknowledged.

Issue I-20 `NoLossCollateralAuction.init` can be called multiple times

Summary

`NoLossCollateralAuction.init` should be a init function. But it can be called multiple times.

Vulnerability Detail

`NoLossCollateralAuction.init` only checks `vaults[vault].calculator`. But it doesn't initialize `vaults[vault].calculator`. Thus, this function can be called multiple time.

```
function init(address vault, address collybus) external override checkCaller {
    if (vaults[vault].calculator != IPriceCalculator(address(0)))
        revert NoLossCollateralAuction__init_vaultAlreadyInit();
    vaults[vault].multiplier = WAD;
    vaults[vault].collybus = ICollybus(collybus);

    emit Init(vault);
}
```

Impact

An init function should only be called once.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/NoLossCollateralAuction.sol#L186-L193

Tool used

Manual Review

Recommendation

`NoLossCollateralAuction.init` should check what it initilaize.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-21 A redundant function in `Collybus`

Summary

There is a redundant function in `Collybus`.

Vulnerability Detail

`Collybus.setParam` seems to be used to set `live`.

```
function setParam(bytes32 param, uint256 data) external override checkCaller {
    if (live == 0) revert Collybus__setParam_notLive();
    if (param == "live") live = data;
    else revert Collybus__setParam_unrecognizedParam();
    emit SetParam(param, data);
}
```

Apparently, it can only be called when `live` is not zero. And the only reasonable choice is set `live` to 0. But it can be done by `Collybus.lock`.

Impact

A redundant function doen't do much harm to the protocol. But Collybus.setParam won't emit `Lock()` log. It could lead to confusions.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Collybus.sol#L74-L79 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Collybus.sol#L181

Tool used

Manual Review

Recommendation

Remove the redundant funtion. Use `Collybus.lock` instead.

Team

The reason both ways to update `live` exist is that generally `lock` is used during Tenebrae when initiating global settlement. Though Collybus may be replaced and shut down on other occasions. This is mostly semantics.

Sherlock

Acknowledged.

Issue I-22 Flash loan amount will be `max` if codex isn't live

Summary

The `maxFlashLoan()` function returns `max` when the codex is no longer live.

Vulnerability Detail

In the scenario where the codex is migrated and the existing one has been deprecated, the `maxFlashLoan()` function will return `max`, potentially giving users the wrong impression that flash loans still work.

Nevertheless, the ERC3156 specification states: The `maxFlashLoan` function MUST return the maximum loan possible for token. If a token is not currently supported `maxFlashLoan` MUST return 0, instead of reverting.

Hence, it can be argued that it shouldn't take into account other factors, like the availability of dependencies (the codex).

Impact

Users may assume flash loan still works since `max` amount is returned.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Flash.sol#L88-L90

Tool used

Manual Review

Recommendation

Return zero if codex is no longer live.

```
- return (token == address(fiat) && locked == 1) ? max : 0;
+ return (token == address(fiat) && locked == 1 && codex.live() != 0) ? max : 0;
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-23 `Publican` can be repeatedly initialized

Summary

The `init()` function of `Publican.sol` can be repeatedly called under certain conditions, causing the Init() event to be spammed.

Vulnerability Detail

The following check is performed to prevent re-initializations:

```
if (v.interestPerSecond != 0) revert Publican__init_vaultAlreadyInit();
```

However, the owner can call `setParam()` to set `v.interestPerSecond` to 0 to call `init()` again.

Impact

Repeated emission (spamming) of the `Init()` event.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L65-L71 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L84

Tool used

Manual Review

Recommendation

Check `v.lastCollected` instead of `v.interestPerSecond` since it cannot be zeroed out after initialization.

```
- if (v.interestPerSecond != 0) revert Publican__init_vaultAlreadyInit();
+ if (v.lastCollected != 0) revert Publican__init_vaultAlreadyInit();
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-24 Do early return if `vaults[vault].lastCollected)` is equal to `block.timestamp`

Summary

If `vaults[vault].lastCollected)` is equal to `block.timestamp`, the `virtualRate()` function can exited earlier.

Vulnerability Detail

In the case where `vaults[vault].lastCollected)=block.timestamp`, the `rate`

```
rate = wmul(
    wpow(
        add(baseInterest, vaults[vault].interestPerSecond),
        sub(block.timestamp, vaults[vault].lastCollected),
        WAD
    ),
    prev
);
```

will be `prev`, so it can be included in the case where `vaults[vault].lastCollected)< block.timestamp`.

Impact

Higher gas costs.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Publican.sol#L117

Tool used

Manual Review

Recommendation

```
- if (block.timestamp < vaults[vault].lastCollected) return prev;
+ if (block.timestamp <= vaults[vault].lastCollected) return prev;
```

Team

Correct. though this method is not used on-chain so gas savings would not be a reason enough to update.

Sherlock

Acknowledged.

Issue I-25 `msg.sender` can be directly used

Summary

Use `msg.sender` instead of assigning to a local memory variable.

Vulnerability Detail

In `settleUnbackedDebt()`, `msg.sender` is assigned to a local variable `debtor`.

```
address debtor = msg.sender;
```

Impact

Higher gas costs.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L451

Tool used

Manual Review

Recommendation

It would be more gas efficient to use `msg.sender` directly.

Team

Intentionally kept to stay close to the original implementation.

Sherlock

Acknowledged.

Issue I-26 Checking of decimals can be omitted

Summary

Checking the equality of `underlierToken` can simply be kept to its address.

Vulnerability Detail

The constructor of `VaultFY.sol` stores the address and decimals of `underlierToken`. Upon initialization, there is a check to ensure that the `fyToken`'s underlier is equivalent to `underlierToken`.

```
if (underlier != underlierToken || 10**IERC20Metadata(fyToken).decimals() !=
↪  underlierScale) {
   revert VaultFY__initialize_invalidUnderlierToken();
}
```

It should be sufficient to check only the address. We can safely assume that the token decimals will be unchanged, because if it could be changed (eg. underlier token is malicious), there's nothing stopping it from changing after initialization.

Impact

Higher gas costs for initialization.

Code Snippet

https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultFY.sol#L85-L87

Tool used

Manual Review

Recommendation

Omit the decimals check.

```
- if (underlier != underlierToken || 10**IERC20Metadata(fyToken).decimals() !=
↪  underlierScale) {
+ if (underlier != underlierToken) {
   revert VaultFY__initialize_invalidUnderlierToken();
}
```

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-27 Missing `tokenId` in `Enter` and `Exit` events of VaultEPT

Summary

The `VaultEPT`'s `Enter` and `Exit` events are missing the `tokenId` fields, which is inconsistent from the other vault contracts.

Vulnerability Detail

The VaultEPT has the following events

```
event Enter(address indexed user, uint256 amount);
event Exit(address indexed user, uint256 amount);
```

which differs from other vaults where the `tokenId` parameter is omitted.

```
event Enter(uint256 indexed tokenId, address indexed user, uint256 amount);
event Exit(uint256 indexed tokenId, address indexed user, uint256 amount);
```

Impact

Subgraph queries can't be made uniform across vaults.

Code Snippet

https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultEPT.sol#L77-L78

Tool used

Manual Review

Recommendation

Add the `tokenId` parameter in the referenced events.

Team

Fix: https://github.com/fiatdao/vaults/pull/9

Sherlock

Fixed.

Issue I-28 Use constants or immutable values to define MAX value

Summary

It's better to use constants or immutable values to define MAX value.

Vulnerability Detail

Hardcode values in lines may result in incorrectly defined values accidentally between functions.

Impact

Incorrectly values may cause inconsistent behaviours.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Collybus.sol#L136 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Flash.sol#L76

Tool used

Manual Review

Recommendation

Define constants / immutables.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-29 Centralized risk of `setParam`

Summary

Centralized risk of `setParam`

Vulnerability Detail

All setParam in this protocol have centralized risk, caller can set parameters without timelock and limit.

Impact

Caller or compromised caller can set parameters immediately and the parameters has no scope limit.

Code Snippet

```
delphi-v2/src/OptimisticOracle.sol
130:     function setParam(

vaults/src/VaultEPT.sol
141:     function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {

vaults/src/VaultFC.sol
153:     function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {

vaults/src/Vault.sol
88:     function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {
236:     function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {

vaults/src/VaultFY.sol
104:     function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {

vaults/src/VaultSY.sol
169:     function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {
180:     function setParam(bytes32 param, uint256 data) external virtual
↪  checkCaller {

fiat/src/Aer.sol
```

```
97:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {
111:    function setParam(bytes32 param, address data) external override
↪  checkCaller {

fiat/src/Codex.sol
170:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {
182:    function setParam(

fiat/src/Publican.sol
78:    function setParam(
93:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {
103:    function setParam(bytes32 param, address data) external override
↪  checkCaller {

fiat/src/Limes.sol
96:    function setParam(bytes32 param, address data) external override
↪  checkCaller {
106:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {
117:    function setParam(
135:    function setParam(

fiat/src/Vault.sol
90:    function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {
238:    function setParam(bytes32 param, address data) external virtual override
↪  checkCaller {

fiat/src/Collybus.sol
74:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {
86:    function setParam(
104:    function setParam(

fiat/src/Flash.sol
73:    function setParam(bytes32 param, uint256 data) external checkCaller {

fiat/src/auctions/CollateralAuction.sol
199:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller checkReentrancy {
214:    function setParam(bytes32 param, address data) external override
↪  checkCaller checkReentrancy {
226:    function setParam(
245:    function setParam(
```


```
fiat/src/auctions/NoLossCollateralAuction.sol
199:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller checkReentrancy {
214:    function setParam(bytes32 param, address data) external override
↪  checkCaller checkReentrancy {
226:    function setParam(
243:    function setParam(

fiat/src/auctions/PriceCalculator.sol
39:    function setParam(bytes32 param, uint256 data) external checkCaller {
92:    function setParam(bytes32 param, uint256 data) external checkCaller {
151:    function setParam(bytes32 param, uint256 data) external checkCaller {

fiat/src/auctions/SurplusAuction.sol
91:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {

fiat/src/auctions/DebtAuction.sol
100:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {

fiat/src/Tenebrae.sol
181:    function setParam(bytes32 param, address data) external override
↪  checkCaller {
195:    function setParam(bytes32 param, uint256 data) external override
↪  checkCaller {
```

Tool used

Manual Review

Recommendation

Use DAO and multisig caller, and add timelock in setParam. Set a limited scope of parameters.

Team

Currently, the DAO has root rights and enforces a 7-8 day delay. Additionally we have https://github.com/fiatdao/guards which define constraints around the range of values and after which time they can be updated by a multisig. Though the multisig currently also has root rights - those will be revoked after the DAO migrations which will performed in the coming weeks.

Sherlock

Acknowledged.

Issue I-30 debtCeiling/debtFloor for uninitialized vault

Summary

Can set debtCeiling/debtFloor for uninitialized vault through setParam.

Vulnerability Detail

Uninitialized vault can set debtCeiling/debtFloor values through setParam.

Impact

It may confuse users who get an uninitialized vault from `codex.vaults[]`.

Code Snippet

https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L188-L189

Tool used

Manual Review

Recommendation

Check vaults should be inited in setParam.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-31 Approving from non-zero allowances

Summary

Approving from non-zero allowances may revert for certain ERC20 tokens.

Vulnerability Detail

Some tokens (Eg. USDT) will revert from approving non-zero to non-zero allowances. It is therefore discouraged to approve from a non-zero allowance.

```
if (IERC20(swapParams.assetIn).allowance(address(this), swapParams.balancerVault)
↪  < underlierAmount) {
   IERC20(swapParams.assetIn).approve(swapParams.balancerVault,
   ↪  type(uint256).max);
```

The occurrence is very low in this case because of the approval to the largest possible value of `type(uint256).max` and would take an extremely long time to utilise such that it falls below `underlierAmount`.

Impact

Loss of functionality.

Code Snippet

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L277-L279

Tool used

Manual Review

Recommendation

None. The issue is merely for informational purposes.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-32 Use latest compiler version

Summary

Use the latest compiler version available.

Vulnerability Detail

For contracts that have not been deployed, it is advisable to use the latest solidity version. Recent compiler versions contain important fixes such as the Head Overflow Bug in Calldata Tuple ABI-Reencoding and inline assembly memory side effects bug.

Impact

Safety from compiler bugs.

Tool used

Manual Review

Recommendation

Upgrade to the latest compiler version available for contracts that are yet to be deployed.

Team

Acknowledged.

Sherlock

Acknowledged.

Issue I-33 Subtraction can be unchecked

Summary

There is an instance where the subtraction can be unchecked.

Vulnerability Detail

In `_decodeNibbles()`, the subtraction of `length` with `skipNibbles` can be unchecked because `length` has been ensured to be greater or equivalent to `skipNibbles`.

Impact

Greater gas costs.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/MerklePatriciaProofVerifier.sol#L266-L269

Tool used

Manual Review

Recommendation

```
- length -= skipNibbles;
+ unchecked { length -= skipNibbles; }
```

Team

This is a valid issue but we would like to postpone this update because of the LUSD3CRV Validator mini-audit that will be next. We will add this change and if needed a few more and we would like to have it rechecked then.

Sherlock

Acknowledged.

Issue I-34 Comment clarifications and typos

Summary

Some comments have spelling / grammar errors, while others could be better phrased for clarity.

Vulnerability Detail

Typographical errors in comments affect clarity and readability. There are also instances where comments could have further description to reduce ambiguity and confusion.

Impact

Code clarity and readability.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L163 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/validators/utils/MerklePatriciaProofVerifier.sol#L102 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Aer.sol#L143-L144 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/Codex.sol#L449 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/VaultEPT.sol#L47-L70 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/vault/VaultEPTActions.sol#L49 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/PriceCalculator.sol#L12 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/PriceCalculator.sol#L62 https://github.com/fiatdao/fiat/blob/9f9bd8dec4f27ce2ccf84d22683b1b313e899f62/src/auctions/PriceCalculator.sol#L122

Tool used

Manual Review

Recommendation

```
- ... already set.
+ ... already unset.

- An Extension/Leaf node is divergent if it "skips" over
+ An Extension/Leaf node is divergent iff it "skips" over
```

```
/// Checks if enough debt exists to be put up for auction
- /// debtAuctionBidSize > (unbackedDebt - queuedDebt - debtOnAuction)
+ /// Reverts if debtAuctionBidSize > (unbackedDebt - queuedDebt - debtOnAuction)

- /// @param debt Amount of debt to settle [wawd]
+ /// @param debt Amount of debt to settle [wad]

- intialization
+ initialization

- uint256 minOutput; // The
+ uint256 minOutput;

- /// Uses LinearDecrease.sol from DSS (MakerDAO) as a blueprint
+ /// Uses LinearDecrease from DSS/abaci.sol (MakerDAO) as a blueprint

- /// Uses StairstepExponentialDecrease.sol from DSS (MakerDAO) as a blueprint
+ /// Uses StairstepExponentialDecrease from DSS/abaci.sol (MakerDAO) as a
↪  blueprint

- /// Uses ExponentialDecrease.sol from DSS (MakerDAO) as a blueprint
+ /// Uses ExponentialDecrease from DSS/abaci.sol (MakerDAO) as a blueprint
```

Team

Fixes in:

- https://github.com/fiatdao/actions/pull/15
- https://github.com/fiatdao/vaults/pull/10
- https://github.com/fiatdao/delphi-v2/pull/14

Sherlock

Fixed.

Issue I-35 Index parameters in events

Summary

Some parameters in events emitted will benefit from being indexed for filtering via subgraphs.

Vulnerability Detail

While indexing parameters will lead to slightly higher gas costs, it would make filtering emitted events easier. This would be handy for devops and subgraph usage.

OptimisticOracle.sol

- `rateId` in `SetParam`, `Unbond` and `ClaimBond` events
- `proposer` in `Bond`, `Unbond` and `ClaimBond` events

Impact

System usability.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L91 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L106-L108

Tool used

Manual Review

Recommendation

```
- event SetParam(bytes32 rateId, bytes32 param, address value);
+ event SetParam(bytes32 indexed rateId, bytes32 param, address value);

- event Bond(address proposer, bytes32[] rateIds);
+ event Bond(address indexed proposer, bytes32[] rateIds);

- event Unbond(address proposer, bytes32 rateId, address receiver);
+ event Unbond(address indexed proposer, bytes32 indexed rateId, address
↪  receiver);
```

```
- event ClaimBond(address proposer, bytes32 rateId, address receiver);
+ event ClaimBond(address indexed proposer, bytes32 indexed rateId, address
↪  receiver);
```

Team

Fixed in PR: https://github.com/fiatdao/delphi-v2/pull/12

Sherlock

Fixed.

Issue I-36 Redundant castings

Summary

Variables are already of the casted type.

Vulnerability Detail

There are instances of variables being unnecessarily casted because they are already of the type to be casted to.

OptimisticOracle.sol:

- `rateConfigs[rateId].validator`

LeverEPTActions.sol:

- `self`

Vault.sol:

- `wad`

Impact

Higher gas costs.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L215 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L264 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L277 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L301 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L318 https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L409

https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L122 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L124 https://github.com/fiatdao/actions/blob/ba925673106997caae5da5a2179208b031995e9d/src/lever/LeverEPTActions.sol#L133

https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/Vault.sol#L124 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/Vault.sol#L257 https://github.com/fiatdao/vaults/blob/d80d9411ee760da01f17f3019f8042a61f505722/src/Vault.sol#L272

Tool used

Manual Review

Recommendation

```
- address(rateConfig.validator)
+ rateConfig.validator

- address(self)
+ self

- int256(wad)
+ wad
```

Team

Casts for the OptimisticOracle fixed in PR: https://github.com/fiatdao/delphi-v2/pull/13

Sherlock

Fixed.

Issue I-37 Unreachable code block in `_push()` from invalid rate type

Summary

The `else` case of `_push()` is unreachable.

Vulnerability Detail

```
} else {
   revert OptimisticOracle__push_invalidRelayerType(rateType);
}
```

will never be entered because `rateType` is of enum type `RateType`. The possible values `rateType` can take has already been handled in the if cases.

Impact

Redundancy.

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L383-L385

Tool used

Manual Review

Recommendation

Remove the referenced lines.

Team

Fixed by PR: https://github.com/fiatdao/delphi-v2/pull/16

Sherlock

Fixed.

Issue I-38 Return `rateIds` directly instead of storing values into and returning another array

Summary

`rateIds` can be returned directly as a gas optimization in `bond()`.

Vulnerability Detail

The `rateIds` array is iterated through and its values are stored into a separate memory local variable `bondedRateIds`. In success cases, `bondedRateIds` is equivalent to `rateIds`. Hence, it would be more gas efficient to directly return `rateIds`.

Impact

Higher gas costs

Code Snippet

https://github.com/fiatdao/delphi-v2/blob/35819264b12d5e204cb7ff18d3317e4aba7c7178/src/OptimisticOracle.sol#L421-L452

Tool used

Manual Review

Recommendation

```
function bond(address proposer, bytes32[] calldata rateIds)
   public
-   returns (bytes32[] memory bondedRateIds)
+   returns (bytes32[] calldata)
{
   ...
-   bondedRateIds = new bytes32[](rateIds.length);
   ...
-      bondedRateIds[i] = rateId;
   ...
-   emit Bond(proposer, bondedRateIds);
+   emit Bond(proposer, rateIds);
+   return rateIds;
}
```

Team

Fixed in https://github.com/fiatdao/delphi-v2/pull/11

Sherlock

Fixed. Return parameter has been removed.