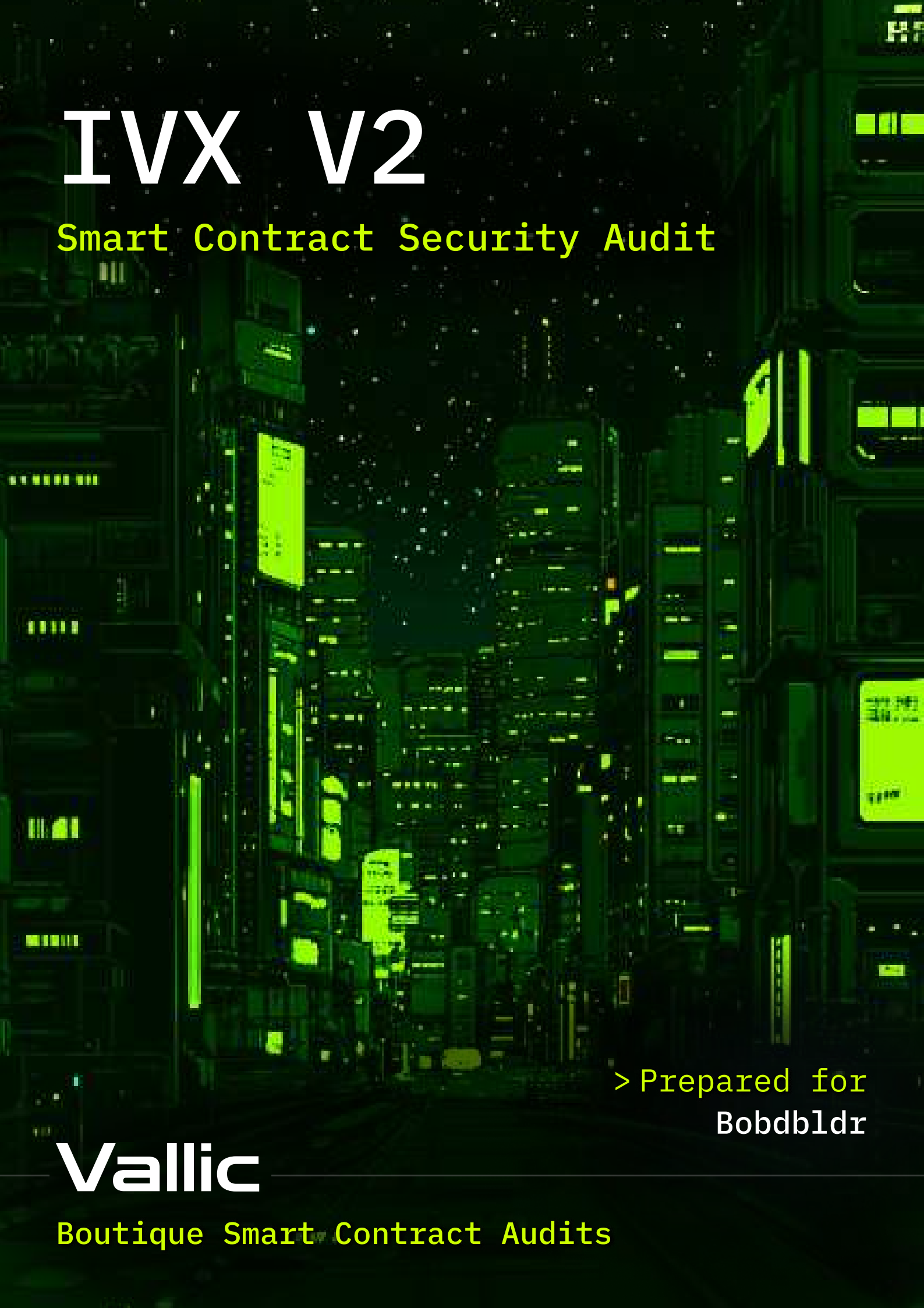

IVX V2

Smart Contract Security Audit

> Prepared for
Bobdbldr

Vallic

Boutique Smart Contract Audits

Contents

 © 2024, Vallic Ltd

About Us

Vallic was founded in 2024 by a team of hackers from the blockchain industry.

We have over a decade of combined expertise within our small team of elite engineers and are recognized as leaders in smart contract development and auditing. Prior to establishing Vallic, our team members worked individually, uncovering critical vulnerabilities in companies such as GitHub and Lucid. Since our inception, we have earned over $400,000 in bug bounties by discovering significant exploits in code and infrastructure for some of the largest companies in the world.

We take each project personally and regularly decline work that we do not believe aligns with our core values. We pride ourselves on being affordable while maintaining exceptional quality and attention to detail. We do not charge excessive fees; the majority of our compensation is derived from findings rather than flat fees. We often collaborate with the same projects repeatedly, at all stages of their development cycles, ensuring that our partner projects launch their code with confidence and the reassurance that it has been reviewed by the best in the industry.

If you are a project looking for a partner team to assist with code security and reviews, please contact us to check availability on our website ↗.

© 2024, Vallic Ltd

1 Summary

Vallic was hired to conduct a security assessment of IVX V2 Protocol between 21st October 2024 and 14th November 2024. During this period, we reviewed the provided code base for security vulnerabilities and documented our findings in this report.

1.1 Classification

Each finding is assessed and assigned an impact rating, which reflects both the severity and likelihood of the issue. The impact is determined based on a combination of factors, including the nature of the vulnerability and its potential exploitability in the context of the client's environment. A highly severe issue with a low likelihood may have a different weight than a less severe but more likely issue. We categorize findings into five impact levels: **CRITICAL**, **HIGH**, **MEDIUM**, **LOW**, and **INFO**.

The following risk classification provides an approximate method for classifying findings which is illustrative of our classification methodology internally, where the impact of a finding is matched with the likelihood of the finding scenario occurring in order to determine the overall severity:

	Impact: **HIGH**	Impact: **MEDIUM**	Impact: **LOW**
Likelihood: **HIGH**	**CRITICAL**	**HIGH**	**MEDIUM**
Likelihood: **MEDIUM**	**HIGH**	**MEDIUM**	**LOW**
Likelihood: **LOW**	**MEDIUM**	**LOW**	**LOW**

Findings which are **CRITICAL** or **HIGH** in severity represent serious vulnerabilities which should be addressed and fixed by engineers. Findings of **MEDIUM** severity may cause minor loss of funds or affect the core protocol functionality in a less severe way. Finally, findings which are **LOW** severity are relatively unlikely or may lead to unexpected behaviour in some functionality, but not significant loss of funds.

 © 2024, Vallic Ltd

1.2 Results

During our assessment, we discovered 28 findings, 4 of which were **CRITICAL** impact and 2 of which were **HIGH** impact. The full count of findings is provided below:

Severity	Count
CRITICAL	4
HIGH	2
MEDIUM	12
LOW	10
Total	28

© 2024, Vallic Ltd

1.3 Summary of Results

ID	Title	Severity	Status
3.1.1	`BrokerRepo`, `ReserveRepo` and `RewardsRepo` Missing Modifier	**CRITICAL**	Resolved
3.1.2	Buy Call Max Reserves Used for Sell Put Positions	**CRITICAL**	Resolved
3.1.3	`_validatePortfolio` Check Errors Preventing `forceClosePositions`	**CRITICAL**	Resolved
3.1.4	Settlement and Liquidation Logic Issue with `_closeAllPositionsContracts`	**CRITICAL**	Resolved

ID	Title	Severity	Status
3.2.1	Swap Amount In Used as Swap Amount Erroneously	**HIGH**	
3.2.2	Settlement Payoff Depends on Spot Price	**HIGH**	Resolved

ID	Title	Severity	Status
3.3.1	`_transferToTreasury` Should Subtract Unclaimed Amount Before Transfer	**MEDIUM**	Resolved
3.3.2	Missing Concatenation in textttgetTokensPriorityOrdered	**MEDIUM**	Resolved
3.3.3	Reserve Calculations Rounded Incorrectly	**MEDIUM**	Resolved
3.3.4	IVLP Transfer Functions Missing Return Values	**MEDIUM**	Resolved
3.3.5	Incorrect Entry Fee Calculation	**MEDIUM**	Resolved
3.3.6	Incorrect Token Out When Swapping Collateral	**MEDIUM**	

 © 2024, Vallic Ltd

ID	Title	Severity	Status
3.3.7	Liquidation Favors the Fee Splitter over the Liquidator	MEDIUM	
3.3.8	Swap Failures Allow for DOS	MEDIUM	
3.3.9	Incorrect Calculation of `minPrice` in Black Scholes Library	MEDIUM	Resolved
3.3.10	`_validateCloseOption` Invalid Validation Causing DOS	MEDIUM	Resolved
3.3.11	Price Feed May Return Stale Prices	MEDIUM	Resolved
3.3.12	`getAmmPnlAndPayOff` Differs from Active P&L Due to Average `p0`	MEDIUM	

ID	Title	Severity	Status
4.1.1	`_distributeFees` Doesn't Validate Distributions	LOW	
4.1.2	Incorrect Condition Used in `_isLiquidatableMMR`	LOW	
4.1.3	Withdraw Should Revert if `canWithdraw` is False	LOW	Resolved
4.1.4	`splitFees` and `collectReceiverFees` Should Use `getWhitelistedTokens`	LOW	Resolved
4.1.5	`claimableRewards` Returns Rewards with Incorrect Decimals	LOW	Resolved
4.1.6	Users Can Force Tokens to not be Removed	LOW	
4.1.7	Unbounded Ceiling of Amounts	LOW	Resolved
4.1.8	Calculating Deviation in `TokenWeightBalancer` Results in Incorrect Results	LOW	
4.1.9	Missing Slippage Protection on Operations	LOW	
4.1.10	Static Fee Used for Swap Pools	LOW	

 © 2024, Vallic Ltd

2 Introduction

2.1 About IVX V2

IVX is a decentralized options Automated Market Maker (AMM) tailored especially for zero days to expiry (0DTE) contracts. The primary focus of IVX is on crypto and real-world assets, such as altcoin options markets, and the protocol aims to provide high leverage exposure of up to 200x to users in a permissionless options market, all through an industry-leading lucid user experience.

2.2 Methodology

During a security audit, we combine automated tools with manual analysis. The process typically involves a thorough code review, supplemented by specialized tools as needed.

Vallic's security assessment focuses on key areas:

1. **Programming Mistakes**: We identify common errors that can lead to serious vulnerabilities, emphasizing manual inspection but also using techniques like fuzz testing, model checking, or formal verification when applicable.
2. **Business Logic**: We analyze the contract's business logic, design documents, and specifications for flaws, inconsistencies, and exploitable risks (e.g., unrealistic economic assumptions, token inflation, arbitrage vulnerabilities). We also verify the code's alignment with the platform's objectives.
3. **Integration Risks**: We review interactions with external data sources, contracts, and oracles, identifying risks from flash loans, price manipulation, and MEV attacks—often overlooked in traditional audits.
4. **Codebase Quality**: We assess adherence to best practices, maintainability, scalability, and optimization for gas efficiency, upgradeability, and decentralization.
5. **Miscellaneous Observations**: We provide non-security-related feedback, such as suggestions for codebase improvements or optimizations, which can enhance overall functionality and efficiency.

 © 2024, Vallic Ltd

3 Detailed Findings

3.1 Critical Severity Findings

3.1.1 `BrokerRepo`, `ReserveRepo` and `RewardsRepo` Missing Modifier

Contract	`BrokerRepo.sol`, `ReserveRepo.sol`, `RewardsRepo.sol`
Likelihood	**HIGH**
Severity	**HIGH**
Impact	**CRITICAL**

Description

`BrokerRepo`, `ReserveRepo` and `RewardsRepo` are missing the `onlySystem` modifier on function calls, this allows any user to update critical state variables and siphon funds.

Recommendations

Add necessary modifiers to the contract setters so they're not permissionless, and instead governed by the `onlySystem` access control modifier.

 © 2024, Vallic Ltd

3.1.2 Buy Call Max Reserves Used for Sell Put Positions

Likelihood	HIGH
Severity	HIGH
Impact	CRITICAL

Description

When calculating the reserves for Sell Put positions, `_reserveSellPut` calls `_calculateMaximumTokenReservations`. During this call, the parameter `_buy` should be passed as False, but it is passed as True erroneously:

```
pragma solidity ^0.8.0;

function _calculateMaximumTokenReservations(
    IWhitelistedTokenRepo _whitelistedTokenRepo,
    IReserveRepo _reserveRepo,
    address _token,
    uint256 _totalSupply,
    bool _buy
) private view returns (uint256 maxBuyReserve) {
    uint256 buyRatio = _whitelistedTokenRepo.getTokenBuyCallRatio(_token);
    uint256 sellRatio = _whitelistedTokenRepo.getTokenSellPutRatio(_token);

    if (_buy) {
        maxBuyReserve = Math.mulDiv(
            _totalSupply,
            buyRatio,
            buyRatio + sellRatio,
            Math.Rounding.Floor
        );
    } else {
        maxBuyReserve = Math.mulDiv(
            _totalSupply,
            sellRatio,
            buyRatio + sellRatio,
            Math.Rounding.Floor
        );
    }
    uint256 _totalUnreservedAmount = _totalSupply -
        _reserveRepo.getTokenTotalReserves(_token);
    maxBuyReserve = Math.min(maxBuyReserve,_totalUnreservedAmount);
}
```

 © 2024, Vallic Ltd

Recommendations

Pass _buy as False when calculating the reserves of an Sell Put position.

 © 2024, Vallic Ltd

3.1.3 `_validatePortfolio` Check Errors Preventing `forceClosePositions`

Contract	`DiemOptions.sol`
Likelihood	**HIGH**
Severity	**HIGH**
Impact	**CRITICAL**

Description

When attempting to force-close positions, the protocol checks if the target portfolio exists using the `_validatePortfolio` check; if not, the call to `forceClosePositions` reverts.

However, the implementation of `_validatePortfolio` has a logic bug where it is inverse. If the portfolio exists, the call reverts, forcing it to never work.

The correct logic should be that, if the portfolio exists, the call should not revert:

```
pragma solidity ^0.8.0;

function _validatePortfolio(
    address _portfolio
) private view returns (address) {
    IPortfolioOrganizer _portfolioOrganizer = IPortfolioOrganizer(
        _getContractAddress(PORTFOLIO_ORGANIZER_CONTRACT)
    );

    if (_portfolioOrganizer.checkPortfolioExistence(_portfolio)) {
        revert SenderHasNoPortfolio(msg.sender);
    }

    _checkPortfolioLiquidation(_portfolio);
    return _portfolio;
}
```

Recommendations

Make the following modifications to the existence check call to ensure that the call reverts correctly:

```
function _validatePortfolio(
    address _portfolio
) private view returns (address) {
```

 © 2024, Vallic Ltd

```
    IPortfolioOrganizer _portfolioOrganizer = IPortfolioOrganizer(
        _getContractAddress(PORTFOLIO_ORGANIZER_CONTRACT)
);

-   if (_portfolioOrganizer.checkPortfolioExistence(_portfolio)) {
+   if (!_portfolioOrganizer.checkPortfolioExistence(_portfolio)) {
        revert SenderHasNoPortfolio(msg.sender);
    }

    _checkPortfolioLiquidation(_portfolio);
    return _portfolio;
```

 © 2024, Vallic Ltd

3.1.4 Settlement and Liquidation Logic Issue with `_closeAllPositionsContracts`

Likelihood	HIGH
Severity	HIGH
Impact	CRITICAL

Description

Both settlement and liquidation mechanisms call the function `_closeAllPositionsContracts`, to close all position contracts and to release that reserves held for that position.

This mechanism closes all position contracts, and then releases the reserves. However, the releasing of the reserves uses `getPositionTotalContracts(_positionId)`. Since the contracts are closed first, this results in an undercount of the position total contracts, `totalNumberOfContracts`.

In the case where the position closed was the only position, `totalNumberOfContracts` will return 0 and the call will revert with a "division by 0 error". This allows a user to inadvertently DOS the contract.

Proof of Concept

The following is a unit test proof-of-concept for the latter case, where the call reverts with a "division by 0 error":

```
pragma solidity ^0.8.0;

function test_SettleDOS() public {
    vm.startPrank(bob);
    address portfolio = portfolioOrganizer.createPortfolio();
    WBTC.transfer(portfolio, 1e8);
    WETH.transfer(portfolio, 1e18);
    diemOptions.openPosition(
        OpenPositionParams({
            token: address(WBTC),
            strikePrice: 65_000e18,
            expireDate: block.timestamp + ONE_DAY,
            positionAction: PositionAction.BUY,
            positionType: PositionType.CALL,
            numberOfContracts: 1_00
        })
    );
    vm.stopPrank();
    vm.warp(block.timestamp + epochRepo.getCurrentEpochEndTimestamp());
    _setAvgPrices();
```

 © 2024, Vallic Ltd

```
    vm.prank(coordinator);
    optionsManager.setEpoch(block.timestamp + ONE_DAY);
    vm.prank(alice);
    vm.expectRevert(stdError.divisionError);
    // panic: division or modulo by zero (0x12)
    settler.settlerPortfolio(portfolio);
}
```

Recommendations

This issue can be fixed by either of the following:

1. Moving the reserve releasing logic in `_closeAllPositionsContracts` to occur prior to closing contracts (in the call to `_closePositionContracts`).
2. Cache `_contracts.optionsRepo.getPositionTotalContracts(_positionId)` prior to closing contracts, and use the cached value when releasing the position reserves after closing contracts.

 © 2024, Vallic Ltd

3.2 High Severity Findings

3.2.1 Swap Amount In Used as Swap Amount Erroneously

Likelihood	HIGH
Severity	MEDIUM
Impact	HIGH

Description

When a swap happens in `_swapPortfolioCollateralTokens`, the protocol expects an `amountToSwap` to be sent to the receiver, and `amountToSwap` is subtracted according to the amount in for the swap. When swapping an X amount of token into Y amount of token out, the USD value of X is $\geq$ USD value of Y, however, the protocol assumes that they're always equal.

A loss will occur but won't be registered/saved, this is because the logic is using the amount in USD as the amount that will be sent to the receiver (in the scenario where the vault is in the `collectLossFromPortfolio` case), the protocol will register that it received X USD (as the amount in value), while it'll receive Y USD (as amount out value), so $X - Y$ is the loss that the vault suffered. However, this loss was not registered in the calculate logic due to the assumption of equality.

Recommendations

`_amountToSwap` should be subtracted according to the USD value of the swapped amount out resulting from the "exact in" swap.

 © 2024, Vallic Ltd

3.2.2 Settlement Payoff Depends on Spot Price

Likelihood	HIGH
Severity	MEDIUM
Impact	HIGH

Description

The settlement payoff of an option post-expiry depends on the current spot price of the underlying. This payoff should be constant and snapshot to the price of the underlying at the point of expiry.

Proof of Concept

The following calculations were performed using the payoff getter function, all numbers are in US dollar equivalents:

	Case 1	Case 2
Average Entry Price	460	460
Average Price	65,000	65,000
Spot Price	50,000	70,000
Strike Price	65,000	65,000
Payoff	-350	-495

 © 2024, Vallic Ltd

3.3 Medium Severity Findings

3.3.1 `_transferToTreasury` Should Subtract Unclaimed Amount Before Transfer

Contract	`RewardsTracker.sol`
Likelihood	**MEDIUM**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

Users can re-calibrate their rewards and claim them later using the `RewardsTracker` contract. These rewards (which are claimable but not yet claimed by the user) are not taken into consideration when transferring the rewards to the treasury during the call to `_transferToTreasury`.

Recommendations

When transferring the rewards to the treasury, the unclaimed claimable rewards should be subtracted from the total balance.

 © 2024, Vallic Ltd

3.3.2 Missing Concatenation in `getTokensPriorityOrdered`

Contract	`TokenWeightBalancer.sol`
Likelihood	**LOW**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

When `getTokensPriorityOrdered` is called for exit in the `TokenWeightBalancer.sol` contract, `_sunsetStableTokens` is computed but never used and is also not concatenated to the returned `_sunsetTokens`.

Recommendations

Concatenate and return the computed sunset stable tokens in `getTokensPriorityOrdered`.

 © 2024, Vallic Ltd

3.3.3 Reserve Calculations Rounded Incorrectly

Contract	`PoolReserver.sol`
Likelihood	**MEDIUM**
Severity	**LOW**
Impact	**MEDIUM**

Description

All reserves calculations should be rounded up, in favor of the protocol, in `_reserveBuyPut`, `_reserveSellPut`, and `_reserveSellCall`:

- PoolReserver.sol#L322-L339 ↗
- PoolReserver.sol#L394-L402 ↗
- PoolReserver.sol#L434-L442 ↗

© 2024, Vallic Ltd

3.3.4 IVLP Transfer Functions Missing Return Values

Contract	`IVLP.sol`
Likelihood	**MEDIUM**
Severity	**LOW**
Impact	**MEDIUM**

Description

IVLP token transfer functions do not return `True` on successful transfers.

This makes the `.transfer` function potentially incompatible with other protocols.

Recommendations

Modify the `transfer` and `transferFrom` functions on the IVLP contract to return their call to `super`:

```
function transfer(address _to, uint256 _value) public override returns (bool){
    _recalibrateRewards(_msgSender());
    _recalibrateRewards(_to);
-   super.transfer(_to, _value);
+   return super.transfer(_to, _value);
}

function transferFrom(address _from, address _to, uint256 _value) public
override returns (bool) {
    _recalibrateRewards(_from);
    _recalibrateRewards(_to);
-   super.transferFrom(_from, _to, _value);
+   return super.transferFrom(_from, _to, _value);
}
```

 © 2024, Vallic Ltd

3.3.5 Incorrect Entry Fee Calculation

Likelihood	LOW
Severity	MEDIUM
Impact	MEDIUM

Description

The user should pay a fixed fee if the amount they add is less than or equal to the target weight. However, if he is adding an amount that makes the real weight strictly greater than the target weight, then more fees should be taken from the user. This is not happening in the call to (`_calculateEntryFees`) due to the calculation of the new fee factor.

Recommendations

Introduce a minimum entry fee that should be paid by all users to avoid this issue.

 © 2024, Vallic Ltd

3.3.6 Incorrect Token Out When Swapping Collateral

Contract	`AccountantResolverLib.sol`
Likelihood	**MEDIUM**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

According to the team and the provided documentation, the remaining collateral tokens are swapped to the most unbalanced token (which in the documentation needs to be stable type, however the team confirmed that this could also be tradable):

- AccountantResolverLib.sol#L137 ↗.

`_collectableTokens` is indeed sorted correctly, but after cutting losses, the balances involved could change, and the swap step could be swapping for an incorrect token:

- AccountantResolverLib.sol#L228 ↗.

Since this logic using the first stable token found, regardless of the balance of that token.

 © 2024, Vallic Ltd

3.3.7 Liquidation Favors the Fee Splitter over the Liquidator

Contract	`PortfolioResolverLib.sol`
Likelihood	**LOW**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

Upon liquidation, fee distribution favors the fee splitter cut over the liquidator fee, which can lower the incentive for liquidation. When liquidating a portfolio whose weight is just over the loss, the liquidator will get nothing in return, as the protocol attempts to fulfill the fee splitter cut first.

- PortfolioResolverLib.sol#L152-L180 ↗

Recommendation

Modify this logic to instead favor the liquidator over the fee splitter, to increase the liquidation incentive.

 © 2024, Vallic Ltd

3.3.8 Swap Failures Allow for DOS

Likelihood	LOW
Severity	MEDIUM
Impact	MEDIUM

Description

The `Swapper.sol` contract executes various swaps and provides a strict deadline and a slippage factor with which to use for executing swaps. These factors, although required, make the swap vulnerable to reverting under certain conditions (such as if the slippage is excessive or the swap has passed its deadline).

However, if a single swap during this process reverts, the entire transaction will revert which makes `collectLossFromPortfolio` and `collectFeesFromPortfolio` vulnerable to a Denial-of-Service whereby every call to execute these swaps fails.

Recommendation

The protocol should allow swaps to fail, by wrapping swaps in a try/catch block instead, and if a particular swap fails the funds should be sent back to the portfolio contract. This will ensure that certain actions, like settlement and liquidation, can't be blocked by this Denial-of-Service. However, we note that in the worst-case scenario this modification allows for some accumulation of debt.

 © 2024, Vallic Ltd

3.3.9 Incorrect Calculation of `minPrice` in Black Scholes Library

Contract	`Pricer.sol`
Likelihood	**LOW**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

`minPrice` is calculated as a modification to spot price by dividing the spot price by `minPriceFactor` without dividing the resulting answer by the basis scale. The answer is also divided by a very high decimal in the call to `multiplyDecimalRoundPrecise`, forcing it to always return 0 or 1.

- Pricer.sol#L290-L292 ↗
- Pricer.sol#L356-L358 ↗

Recommendation

Modify the calculation of `minPrice` to the following:

```
uint256 minPrice = ((spotPrecise * optionParams.minPriceFactor) /
    BASIS_POINTS_DIVISOR).preciseDecimalToDecimal();
```

 © 2024, Vallic Ltd

3.3.10 `_validateCloseOption` Invalid Validation Causing DOS

Contract	`DiemOptions.sol`
Likelihood	**LOW**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

In the `DiemOptions.sol` contract, the call to `_validateCloseOption` checks if the number of contracts a user is closing is less than the minimum number of contracts:

- DiemOptions.sol#L763-L768 ↗

However, this validation check is incorrect because it blocks users who have positions opened with the minimum number of contracts from closing their positions if the co-ordinator increases the minimum number of contracts after they open their positions initially.

Recommendation

Remove that validation, and allow the user to close as many contracts as they want up to their position size. The validation should only be done on the remaining contracts in the position.

 © 2024, Vallic Ltd

3.3.11 Price Feed May Return Stale Prices

Contract	`PriceFeed.sol`
Likelihood	**LOW**
Severity	**MEDIUM**
Impact	**MEDIUM**

Description

`getTokenPrice` may return stale prices for tokens since it does not implement checks on the prices returned.

Recommendation

This call should add some additional checks to ensure safety:

- Check the `updated.at` of the latest round data, since each feed has a different stale period.
- Check if the price to be returned is greater than 0.

© 2024, Vallic Ltd

3.3.12 `getAmmPnlAndPayOff` Differs from Active P&L Due to Average `p0`

Likelihood	LOW
Severity	MEDIUM
Impact	MEDIUM

Description

`getAmmPnlAndPayOff` may differ from the active positions P&L sum, because of the use of the average price, `p0`, which affects the minted shares when depositing into the `Broker.sol` contract.

 © 2024, Vallic Ltd

4 Discussion

4.1 Low Severity Findings

The following section contains findings from our review which were classified as **LOW** severity by the team. These can be considered observational, and our auditors did not believe that they represented substantial security threats to the protocol itself. These findings are compiled and provided here for completeness.

4.1.1 `_distributeFees` Doesn't Validate Distributions

Contract	`FeeSplitter.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description

The call to `_distributeFees`, loops over the length of the distribution, however, it doesn't check if there are distributions in the first place.

In case of no distributions, any transaction that targets the `_distributeFees` function will revert with an underflow error.

Recommendation

A zero-length check should be added before looping over the distributions.

 © 2024, Vallic Ltd

4.1.2 Incorrect Condition Used in `_isLiquidatableMMR`

Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description

According to the documentation, a portfolio is liquidatable if MMR rises above some threshold.

However, according to `_isLiquidatableMMR`, a portfolio can be liquidated if its MMR is greater or equal to the threshold, contradicting the docs.

 © 2024, Vallic Ltd

4.1.3 Withdraw Should Revert if `canWithdraw` is False

Contract	`PortfolioOrganizer.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description/Recommendation

The call to `withdraw` should revert if `canWithdraw` is false, following the ”fail-early and fail-loud” convention.

 © 2024, Vallic Ltd

4.1.4 splitFees & collectReceiverFees Should Use getWhitelistedTokens

Contract	FeeSplitter.sol
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description/Recommendation

In the Feesplitter.sol contract, calls to splitFees and collectReceiverFees should use calls to getWhitelistedTokens instead of getActiveTokens, as active tokens don't contain sunset tokens, whilst sunset tokens can still accumulate fees (such as on withdrawals).

 © 2024, Vallic Ltd

4.1.5 `claimableRewards` Returns Rewards with Incorrect Decimals

Contract	`RewardsTracker.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description

`RewardsTracker::claimableRewards` returns rewards with incorrect decimals, where the answer is not rounded to the precise reward token's decimals.

 © 2024, Vallic Ltd

4.1.6 Users Can Force Tokens to not be Removed

Contract	`Pool.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description

In the `Pool.sol` contract, after the token has finished the sunset state and the admin decides to remove the token, it will check if the balance is 0 to be able to remove it:

- Pool.sol#L302-L305 ↗

Since we are getting the value by checking the balance, an attacker can send 1 wei to the `REWARDS_VAULT_CONTRACT` causing it always to revert whenever the admin tries to remove the token.

 © 2024, Vallic Ltd

4.1.7 Unbounded Ceiling of Amounts

Contract	`AccountantResolverLib.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description

All ceiling computation of amounts should be bounded to the portfolio's balance, so the transaction doesn't revert unexpectedly:

- AccountantResolverLib.sol#L93-L98 ↗
- AccountantResolverLib.sol#L306-L311 ↗
- AccountantResolverLib.sol#L395-L400 ↗

© 2024, Vallic Ltd

4.1.8 Calculating Deviation in `TokenWeightBalancer` Results in Incorrect Results

Contract	`TokenWeightBalancer.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description

Calculating the deviation in `TokenWeightBalancer` whilst depending on the entry/exit will sometimes result in incorrect results. This is because the algorithm used does not always converge on the best options.

 © 2024, Vallic Ltd

4.1.9 Missing Slippage Protection on Operations

Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description/Recommendation

Slippage protection should be added to give users higher control over their actions, particularly on the deposit, withdrawal and open position functions.

 © 2024, Vallic Ltd

4.1.10 Static Fee Used for Swap Pools

Contract	`Swapper.sol`
Likelihood	**LOW**
Severity	**LOW**
Impact	**LOW**

Description/Recommendation

In `Swapper.sol`, use a preset pool fee for the pair, and have then use 3000 as the fallback/default value:

- Swapper.sol#L85 ↗

 © 2024, Vallic Ltd

5 Disclaimer

This security assessment was conducted with the aim of identifying potential vulnerabilities within the specified scope. However, Vallic makes no guarantees regarding the discovery of every possible issue. As such, the findings in this report should not be interpreted as ensuring the absence of any future vulnerabilities, whether within the reviewed code or introduced in subsequent versions.

Additionally, Vallic is not responsible for any code or modifications added to the project after the assessment, and these results should not be considered a comprehensive review. For this reason, we recommend that security audits be performed regularly, complemented by independent reviews and bug bounty programs to address any evolving risks.

For each identified issue, Vallic provides suggested mitigations, including code examples where applicable. These suggestions are meant to illustrate possible solutions and are not exhaustive or guaranteed to be error-free. We encourage our clients to treat these recommendations as a starting point for further investigation and refinement. We are available for additional consultation if further guidance or clarification is needed.

Lastly, please note that the contents of this report are for informational purposes only. It should not be construed as legal, financial, tax, or investment advice. Vallic does not endorse or solicit any particular project or entity, and nothing in this report should be considered as a recommendation or an official endorsement.

 © 2024, Vallic Ltd