LEDGEROPS

Stabl.Fi Rebase

Smart Contract Audit

EXECUTIVE SUMMARY

LedgerOps was engaged by the Stabl.Fi team to perform a source code audit of their rebasing smart contracts.

These findings are based on commit: **5fe1aca8e33644d33ecdcd537264057e53383a92** in the GitHub repository shared with us (https://github.com/StablFi/stabl-contracts-audit).

At a high level, the CASH and Vault smart contracts enable Stabl.Fi to allow users to invest stablecoins into a token that automatically generates yield and distributes it back to the users through six primary contracts:

- **VaultAdmin.sol** – Contains functionality regarding the administration of the Vault.
- **VaultCore.sol** – Contains the core minting, investing, and redeeming functions of the Vault.
- **CASH.sol** – The ERC-20 compatible native token of Stabl.Fi that automatically distributes yield back to holders.
- **Harvester.sol** – Responsible for harvesting yield from each strategy, collecting fees, and distributing the yield to the Dripper.
- **Dripper.sol** – Distributes yield to the Vault smoothly (time-averaged fashion).

LedgerOps has identified two (2) high severity findings, two (2) findings of moderate severity, three (3) findings of low severity, and five (5) informational findings.

Amongst the high severity findings, a common theme was present – issues related to merging existing code with new operating paradigms and security objectives. LedgerOps identified several key scenarios where the key functionality was changed and resulted in unexpected situations:

- Changed expected invocation of a function would have broken key functionality.
- New yield strategies could potentially enable an economic attack.

The moderate findings include an apparent departure from expected functionality and a centralization risk due to new capabilities added to the Strategist role. LedgerOps also encountered several low or informational findings due to the nascent development of the platform. LedgerOps encourages Stabl.Fi to implement

thorough testing on all modified features and conduct security testing against the rest of the platform, as the scope of this engagement was limited to changes in the core functionality of the Vault, yield harvesting, and native token.

LedgerOps greatly appreciates the opportunity to work with the Stabl.Fi development team and looks forward to supporting the team throughout the entire Software Development Life Cycle. The team was very responsive and proactive in answering any questions from the security testers and changing instances of discovered issues.

The discovered issues are highlighted below along with their respective severity:

Finding Name	Finding Severity	Status
harvestAndDistribute onlyGovernor Modifier Breaks Payout Functionality	**High**	**Remediated**
Potential Economic Attack Vulnerabilities	**High**	**Remediated**
Payout Function Does Not Pay msg.sender	**Medium**	**Remediated**
Strategist Role Poses a Potential Centralization Risk	**Medium**	**Remediated**
OGN Buyback Functionality Included in Allocate	**Low**	**Remediated**
redeemFee Calculation Does Not Check redeemFeeBps	**Low**	**Remediated**

Code Sanitization: Unneeded Code from Predecessors Should Be Removed; Additional Documentation and Testing Specific to CASH/Stabl Should Be Added	**Low**	**In Progress**
Redemption Calculation Unevenly Redeems from Strategies Earliest in Array	**Informational**	**Risk Accepted**
Lab and Team Fees Distributed on Both Payout and Redeem	**Informational**	**Remediated**
_calculateRedeemOutput() Contains Unnecessary Calculations	**Informational**	**Risk Accepted**
VaultAdmin._balance() Shadows a Storage Variable Name	**Informational**	**Remediated**
Function justMint Should be Removed from VaultCore	**Informational**	**Remediated**

CONTENTS

METHODOLOGY

LedgerOps uses proprietary tools and testing practices combining manual and automated capabilities to develop a tailored, accurate, and highly optimized process. Our methodology helps uncover logic, implementation, and structural flaws that do not follow the industry's best practices. Our wide variety of testing includes dynamic analysis, static analysis, governance review, economic risks, and line-by-line code review.

Our team consists of some of the best security researchers in the industry. LedgerOps aims to provide continuous, qualitative results. Secure development practices and auditing engagements have proved critical as projects introduce new functionalities into their ecosystem. Our mission is to make the Web3 ecosystem a safer place and challenge the entire Web3 security industry to do the same.

SUMMARY OF FINDINGS

Vulnerabilities

High	Medium	Low	Informational
2	2	3	5

Vulnerability Details

#	Title	Type	Severity
1	harvestAndDistribute onlyGovernor Modifier Breaks Payout Functionality	Authorization Controls	High
2	Potential Economic Attack Vulnerabilities	Economic Attack	High

3	Payout Function Does Not Pay msg.sender	Code Completeness	Medium
4	Strategist Role Poses a Potential Centralization Risk	Centralization Risk	Medium
5	OGN Buyback Functionality Included in Allocate	Deprecated Functionality	Low
6	redeemFee Calculation Does Not Check redeemFeeBps	Improper Validation	Low
7	Code Sanitization: Unneeded Code from Predecessors Should Be Removed; Additional Documentation and Testing Should Be Added	Code Completeness	Low
8	Redemption Calculation Unevenly Redeems from Strategies Earliest in Array	Code Style	Informational
9	Lab and Team Fees Distributed on Both Payout and Redeem	Code Completeness	Informational
10	_calculateRedeemOutput() Contains Unnecessary Calculations	Resource Conservation	Informational
11	VaultAdmin._balance() Shadows a Storage Variable Name	Code Style	Informational
12	Function justMint Should be Removed from VaultCore	Active Debug Code	Informational

harvestAndDistribute onlyGovernor Modifier Breaks Payout Functionality

Severity	Level of Effort	Type
High	Low	Authorization Controls

Issue Description

harvestAndDistribute has the modifier onlyGovernor, but the Vault's _payout function is currently designed to call harvestAndDistribute.

Impact

The Governor address will have to call harvestAndDistribute directly on a regular basis; payout functionality is broken for all other users.

Issue Remediation

Change the modifier to onlyVaultOrGovernor.

Supporting Evidence

Location: contracts/harvest/Harvester.sol:Harvestor.sol:144

```
144    function harvestAndDistribute() external onlyGovernor nonReentrant {
145        _harvest();
146        _distribute();
147    }
```

Stable.Fi Remediation

Modifier changed to onlyVaultOrGovernor

```
    function harvestAndDistribute() external onlyVaultOrGovernor nonReentrant {
        _harvest();
        _distribute();
    }
```

References

- N/A

Potential Economic Attack Vulnerabilities

Severity	Level of Effort	Type
High	Moderate	Economic Attack

Issue Description

If any of the strategies' value calculations can be externally influenced and changed within a block, an economic attack could be leveraged to extract funds from the Vault.

The root of this issue is that the Vault's total balance is calculated by immediately requesting the value of each of the strategies at a single point in time, which could change throughout a transaction (or block). The value calculation of a strategy may include using a price oracle to calculate the value of assets within the strategy. If the price oracles used can be manipulated to return different values during a transaction or block, the appearance of profit could enable an economic attack where the Vault's calculations show that the value has gone up, but the assets have not been liquidated, and the profit has not yet been realized.

Though a complete audit of the strategies used was not in scope, it appears that some strategies' checkBalance functions external functionality to calculate the prices of related assets. Suppose any strategy's checkBalance functionality can be manipulated by an attacker within a transaction or block (such as by borrowing a tremendous amount from a liquidity pool using flash loans). In that case, it is a serious threat as it could allow an attacker to manipulate the Vault's perception of value and extract funds. An attack of this nature was conducted against a similar system and is documented at the following link:
https://medium.com/harvest-finance/harvest-flashloan-economic-attack-post-mortem-3cf900d65217

Impact

If an attacker can manipulate the price of a strategy within a transaction. In that case, they could mint a large amount of CASH, then begin price manipulation, trigger a rebase to cause a distribution of fake profit, and finally redeem their initial deposit plus their percentage of the fake profit. The result is a transfer of stablecoin to the attacker, and since the value of CASH cannot be lowered by a rebase, the

ratio of stablecoin backing to CASH will be affected. The amount an attacker can redeem is limited by the maxSupplyDiff check in VaultCore._redeem, but this is not fully effective since the backing value and cash supply both include the attacker's deposit. Additionally, attackers could repeat this attack multiple times.

Supporting Evidence

VaultCore._checkBalance and checkBalance of all strategies

```
function _checkBalance()
    internal
    view
    virtual
    returns (uint256 balance)
{
    IERC20 asset = IERC20(primaryStableAddress);
    balance = asset.balanceOf(address(this));

    for (uint256 i = 0; i < allStrategies.length; i++) {
        IStrategy strategy = IStrategy(allStrategies[i]);
        console.log("Checking Balance of ", allStrategies[i]);
        balance = balance.add(strategy.checkBalance());
    }
}
```

Issue Remediation

Examine all strategies' checkBalance functions and ensure that an attacker cannot manipulate the price oracles and value calculations to change within a single block. If enforcing this requirement on strategies is not possible, a vault redesign should be considered similar to the mitigation strategies discussed in the Harvest Finance post-mortem reference linked below. It is imperative to consider that attackers could bundle multiple transactions using FlashBots. In the future (post Merge), scenarios may arise that enable attacks that span across two or three contiguous blocks.

Stable.Fi Remediation

```
    function checkBalance()
        public
        view
        override
        returns (uint256)
    {
        // LP tokens in this contract. This should generally be nothing as we
        // should always stake the full balance in the Gauge, but include for
        // safety
        uint256 primaryStableBalance = primaryStable.balanceOf(address(this));
        uint256 balance;
        (, , uint256 totalPTokens) = _getTotalPTokens();
        // console.log("Am3Curve - checkBalance - totalPTokens: ", totalPTokens);
        ICurvePool curvePool = ICurvePool(platformAddress);
        if (totalPTokens > 0) {
            balance = curvePool.calc_withdraw_one_coin(totalPTokens,int128(uint128(_getCoinIndex(address(primaryStable)))));
            // console.log("Am3Curve - checkBalance - primaryToken: ", balance);
        }
        return primaryStableBalance + balance;
    }

```

Figure 1: Am3CurveStrategy

```
    }
    function checkBalance()
        external
        view
        override
        returns (uint256 balance)
    {

        uint256 primaryStableBalance = primaryStable.balanceOf(address(this));
        // console.log("primaryStableBalance", primaryStableBalance);
        uint256 userLPBalance = dodoMine.balanceOf(address(this));
        // console.log("userLPBalance", userLPBalance);
        if (userLPBalance > 0) {
            uint256 usdcLPTokenTotalSupply = usdcLPToken.totalSupply();
            // console.log("usdcLPTokenTotalSupply", usdcLPTokenTotalSupply);
            (uint256 baseTarget,) = dodoV1UsdcUsdtPool.getExpectedTarget();
            // console.log("baseTarget", baseTarget);
            uint256 primaryStableAmount = baseTarget * userLPBalance / usdcLPTokenTotalSupply;
            // console.log("primaryStableAmount", primaryStableAmount);
            primaryStableBalance += primaryStableAmount - (primaryStableAmount * 6 / 10000); // - 0.06 %
        }
        return primaryStableBalance;
    }
    function supportsAsset(address _asset)
        external
        view
        override
        returns (bool)
    {
        return ( (_asset == address(primaryStable)) && (_asset == assetsMapped[0] ) );
    }
```

Figure 2: DodoStrategy

```
function checkBalance()
    external
    view
    override
    returns (uint256)
{
    uint256 primaryStableBalance = primaryStable.balanceOf(address(this));
    uint256 token0Balance;
    uint256 lpTokenBalance = meshSwapToken0.balanceOf(address(this));
    // console.log("lpTokenBalance:", lpTokenBalance);

    // TODO: MoreClean workground for handling non-six decimal token0
    // Fix for handling token0 with 18 decimals
    if (IERC20Metadata(address(token0)).decimals() != IERC20Metadata(address(primaryStable)).decimals()) {
        lpTokenBalance = lpTokenBalance.div(10 ** (IERC20Metadata(address(token0)).decimals() - IERC20Metadata(address(primaryStable)).decimals())); // e12 = e18 - e6
    }
    // console.log("lpTokenBalance:", lpTokenBalance);
    if (lpTokenBalance > 0) {
        uint256 exchangeRateStored = meshSwapToken0.exchangeRateStored();
        // console.log("exchangeRateStored:", exchangeRateStored);
        token0Balance = exchangeRateStored.mul(lpTokenBalance).div(1e18);
    }
    // console.log("token0Balance:", token0Balance.scaleBy(IERC20Metadata(address(token0)).decimals() , 6 ));
    uint256 primaryStableBalanceFromToken0;
    if ( (address(token0) != address(primaryStable))  ) {
        if (token0Balance > 0) {
            primaryStableBalanceFromToken0 = onSwap(
                balancerVault,
                poolId,
                IVault.SwapKind.GIVEN_IN,
                token0,
                primaryStable,
                token0Balance.scaleBy(IERC20Metadata(address(token0)).decimals() , 6 )
            );
            // console.log("Token0 swap -  primaryStableBalanceFromToken0 ", primaryStableBalanceFromToken0);
        }
    } else {
        primaryStableBalanceFromToken0 += token0Balance;
    }
    return primaryStableBalanceFromToken0 + primaryStableBalance;
}
```

Figure 3: DystopiaStrategy

```
function checkBalance()
    external
    view
    override
    returns (uint256 balance)
{
    uint256 primaryStableBalance = primaryStable.balanceOf(address(this));
    uint256 token0Balance = token0.balanceOf(address(this));
    uint256 token1Balance = token1.balanceOf(address(this));

    // Fetch amount of penPool LP currently staked
    address userProxyThis = penLens.userProxyByAccount(address(this));
     // console.log("dystPair", address(dystPair));
    address stakingAddress = penLens.stakingRewardsByDystPool(address(dystPair));
     // console.log("stakingAddress: ", stakingAddress);
    uint256 lpTokenBalance = IERC20(stakingAddress).balanceOf(userProxyThis);
    lpTokenBalance += gauge.balanceOf(address(this));
    if (lpTokenBalance > 0) {
        uint256 totalLpBalance = dystPair.totalSupply();
        (uint256 reserve0, uint256 reserve1) = getReserves();
        token0Balance += reserve0 * lpTokenBalance / totalLpBalance;
        token1Balance += reserve1 * lpTokenBalance / totalLpBalance;
    }
    // console.log("tokenBalance", token0Balance, token1Balance);

    uint256 primaryStableBalanceFromToken0;
    if ( (address(token0) != address(primaryStable))  ) {
        if (token0Balance > 0) {
            primaryStableBalanceFromToken0 = onSwap(
                balancerVault,
                poolId,
                IVault.SwapKind.GIVEN_IN,
                token0,
                primaryStable,
                token0Balance
            );
            // console.log("Token0 swap -  primaryStableBalanceFromToken0 ", primaryStableBalanceFromToken0);
        }
    } else {
        primaryStableBalanceFromToken0 += token0Balance;
    }

    uint256 primaryStableBalanceFromToken1;
    if ( (address(token1) != address(primaryStable))  ) {
        if (token1Balance > 0) {
            primaryStableBalanceFromToken1 = onSwap(
                balancerVault,
                poolId,
                IVault.SwapKind.GIVEN_IN,
                token1,
                primaryStable,
                token1Balance
            );
            // console.log("Token1 swap -  primaryStableBalanceFromToken1 ", primaryStableBalanceFromToken1);
        }
    } else {
        primaryStableBalanceFromToken1 += token1Balance;
    }
     // console.log("primaryStableBalanceFromToken0: ", primaryStableBalanceFromToken0);
     // console.log("primaryStableBalanceFromToken1: ", primaryStableBalanceFromToken1);
    return primaryStableBalanceFromToken0 + primaryStableBalanceFromToken1 + primaryStableBalance;
}
```

Figure 4: MeshSwapStrategy

```
    function checkBalance()
        external
        view
        override
        returns (uint256)
    {
        return _totalValue(false);
    }

    function _collectRewards() internal {
         // console.log("Starting collection of rewards");
        // claim rewards
        meshSwapPair.claimReward();
         // console.log("claimStakingRewards called");
        // sell rewards
        uint256 totalUsdc;
        uint256 meshBalance = meshToken.balanceOf(address(this));
        if (meshBalance > 10 ** 13) {
            uint256 meshUsdc = _swapExactTokensForTokens(
                address(meshToken),
                address(primaryStable),
                meshBalance,
                address(this)
            );
            totalUsdc += meshUsdc;
        } else {
             // console.log("Not enough mesh tokens to sell");
        }
        uint256 balance = primaryStable.balanceOf(address(this));
        if (balance > 0) {
            emit RewardTokenCollected(
                harvesterAddress,
                address(primaryStable),
                balance
            );
            primaryStable.transfer(harvesterAddress, balance);
        }
    }
```

Figure 5: MeshSwapStrategyDual

```
    function checkBalance()
        external
        view
        override
        returns (uint256)
    {
        uint256 token0Balance = token0.balanceOf(address(this));
        uint256 token1Balance = token1.balanceOf(address(this));

        uint256 lpTokenBalance = quickSwapPairStaker.balanceOf(address(this));
        if (lpTokenBalance > 0) {
            uint256 totalLpBalance = quickSwapPair.totalSupply();
            (uint256 reserve0, uint256 reserve1,) = quickSwapPair.getReserves();
            token0Balance += reserve0 * lpTokenBalance / totalLpBalance;
            token1Balance += reserve1 * lpTokenBalance / totalLpBalance;
        }

        uint256 primaryStableBalanceFromToken0;
        if ( (address(token0) != address(primaryStable))  ) {
            if (token0Balance > 0) {
                primaryStableBalanceFromToken0 = onSwap(
                    balancerVault,
                    poolId,
                    IVault.SwapKind.GIVEN_IN,
                    token0,
                    primaryStable,
                    token0Balance
                );
                // console.log("Token0 swap -  primaryStableBalanceFromToken0 ", primaryStableBalanceFromToken0);
            }
        } else {
            primaryStableBalanceFromToken0 += token0Balance;
        }

        uint256 primaryStableBalanceFromToken1;
        if ( (address(token1) != address(primaryStable))  ) {
            if (token1Balance > 0) {
                primaryStableBalanceFromToken1 = onSwap(
                    balancerVault,
                    poolId,
                    IVault.SwapKind.GIVEN_IN,
                    token1,
                    primaryStable,
                    token1Balance
                );
                // console.log("Token1 swap -  primaryStableBalanceFromToken1 ", primaryStableBalanceFromToken1);
            }
        } else {
            primaryStableBalanceFromToken1 += token1Balance;
        }
        // console.log("primaryStableBalanceFromToken0: ", primaryStableBalanceFromToken0);
        // console.log("primaryStableBalanceFromToken1: ", primaryStableBalanceFromToken1);
        return primaryStableBalanceFromToken0 + primaryStableBalanceFromToken1;
    }
```

Figure 6: QuickSwapStrategy

```
    function checkBalance()
        external
        view
        override
        returns (uint256)
    {
        uint256 tokenIndex = synapsePool.getTokenIndex(address(primaryStable));
        uint256 primaryStableBalance = primaryStable.balanceOf(address(this));

        (uint256 amount,) = synapseStakerPool.userInfo(synapseStakerPoolId, address(this));
        if (amount > 0) {
            primaryStableBalance += synapsePool.calculateRemoveLiquidityOneToken(amount, uint8(tokenIndex));
        }
        return primaryStableBalance;
    }
    function supportsAsset(address _asset)
        external
        view
        override
        returns (bool)
    {
        return _asset == address(primaryStable);
    }
    /* NOT NEEDED */
    function safeApproveAllTokens() external override  {}
    function _abstractSetPToken(address _asset, address _cToken)internal override {}
}
```

Figure 7: SynapseStrategy

References

- https://medium.com/harvest-finance/harvest-flashloan-economic-attack-post-mortem-3cf900d65217
- https://samczsun.com/so-you-want-to-use-a-price-oracle/

Payout function Does Not Pay msg.sender

Severity	Level of Effort	Type
Medium	Low	Code Completeness

Issue Description

The VaultAdmin.payout() appears designed to replace the functionality of Harvester.harvestAndSwap(), but no payment is sent to msg.sender, which means there is no incentive for users to trigger the harvest functionality.

Impact

The OUSD docs describe an incentivized harvest structure where the user that calls harvestAndSwap gets 1% of the proceeds, thus creating an incentive to trigger harvests. Without such an incentive, the system will require continual interaction and expenditure from the Governor or other invested stakeholders to trigger harvests, which could decrease user interest.

Supporting Evidence

VaultAdmin.sol:521-528

```
    function payout() external {
        _payout();
    }
    ftrace | funcSig
    function _payout() internal {
        IHarvester(harvesterAddress).harvestAndDistribute();
        IDripper(dripperAddress).collectAndRebase();
        _balance();
    }
```

Issue Remediation

Calculate and transfer a percentage of the harvest to msg.sender in _payout by having harvestAndDistribute() return the amount distributed to the vault.

Stable.Fi Remediation
Team Response: Decided to keep this design and call the function as needed using Gelato or a similar construct.

References

- https://github.com/OriginProtocol/origin-dollar/blob/f0a2ce23663d32a8734d8f90b8f903ff66402961/contracts/contracts/harvest/Harvester.sol#L259

Strategist Role Poses Potential Centralization Risk

Severity	Level of Effort	Type
Medium	Low	Centralization Risk

Issue Description

The strategist role in OUSD exists to allow low-touch maintenance and enable faster disaster-recovery actions, but it also lowers the bar for an attacker to be able to compromise a protocol's governance and be able to access administrative functionality. In the case of the Strategist role, several functions would allow an attacker to effectively compromise the system if they can compromise the Strategist role.

Impact

A compromised Strategist could completely disrupt the system or steal all of the funds, depending on the function.

- VaultAdmin.setStrategyWithWeights: the storage variable strategyWithWeight is set without checking input addresses are in allStrategies and strategies[addr].isSupported is true. When combined with the ability to call VaultAdmin.balance() which is also marked onlyGovernorOrStrategist, a compromised Strategist could transfer all holdings to an arbitrary address

- VaultAdmin.setQuickDepositStrategies: the storage variable quickDepositStrategies is set with a list of unchecked addresses, which will be the default recipients for all future assets deposited via mint and distributed by _quickAllocate , which does not check that the addresses in quickDepositStrategies have been approved.

- VaultAdmin.setSwapper: this sets addresses that are used by strategies for asset swaps and will be used to convert all assets deposited via VaultCore.mint by way of VaultCore._swapAsset().

- VaultAdmin.setPrimaryStable: this changes the primary asset used for vault valuation and is a linchpin variable.

Supporting Evidence

VaultAdmin.sol: setStrategyWithWeights, setQuickDepositStrategies, setSwapper, setPrimaryStable

```
function setStrategyWithWeights(StrategyWithWeight[] calldata _strategyWithWeights) external onlyGovernorOrStrategist
    uint256 totalTarget = 0;
    for (uint8 i = 0; i < _strategyWithWeights.length; i++) {
        StrategyWithWeight memory strategyWithWeight = _strategyWithWeights[i];
        require(strategyWithWeight.strategy != address(0), "weight without strategy");
        require(
            strategyWithWeight.minWeight <= strategyWithWeight.targetWeight,
            "minWeight shouldn't higher than targetWeight"
        );
        require(
            strategyWithWeight.targetWeight <= strategyWithWeight.maxWeight,
            "targetWeight shouldn't higher than maxWeight"
        );
        totalTarget += strategyWithWeight.targetWeight;
    }
```

```
function setQuickDepositStrategies(address[] calldata _quickDepositStrategies) external onlyGovernorOrStrategist
    quickDepositStrategies = _quickDepositStrategies;
}
```

```
function setSwapper(address _balancerVault, bytes32 _balancerPoolId) external onlyGovernorOrStrategist
    balancerVault = _balancerVault;
    balancerPoolId = _balancerPoolId;
}
```

```
function setPrimaryStable(address _primaryStable) external onlyGovernorOrStrategist {
    primaryStableAddress = _primaryStable;
}
```

Issue Remediation

Use the onlyGovernor modifier for the functions listed above and check that all incoming strategy addresses have been approved before assigning them to a storage variable.

Stable.Fi Remediation

Team Response: The recommended fixes have been implemented.

```
function setStrategyWithWeights(StrategyWithWeight[] calldata _strategyWithWeights) external onlyGovernor {
    uint256 totalTarget = 0;
    for (uint8 i = 0; i < _strategyWithWeights.length; i++) {
        StrategyWithWeight memory strategyWithWeight = _strategyWithWeights[i];
        require(strategies[strategyWithWeight.strategy].isSupported, "Strategy should be supported by the Vault")
        require(strategyWithWeight.strategy != address(0), "Weight without strategy");
        require(
            strategyWithWeight.minWeight <= strategyWithWeight.targetWeight,
            "minWeight shouldn't higher than targetWeight"
        );
        require(
            strategyWithWeight.targetWeight <= strategyWithWeight.maxWeight,
            "targetWeight shouldn't higher than maxWeight"
        );
        totalTarget += strategyWithWeight.targetWeight;
    }
```

```
function setQuickDepositStrategies(address[] calldata _quickDepositStrategies) external onlyGovernor {
    for (uint8 i = 0; i < _quickDepositStrategies.length; i++) {
        require(strategies[_quickDepositStrategies[i]].isSupported, "Strategy should be supported by the Vault")
    }
    quickDepositStrategies = _quickDepositStrategies;
}
```

```
function setSwapper(address _balancerVault, bytes32 _balancerPoolId) external onlyGovernor {
    require(_balancerVault != address(0), "Empty Swapper Address");
    require(_balancerPoolId != "", "Empty pool id not allowed");
    balancerVault = _balancerVault;
    balancerPoolId = _balancerPoolId;
}
```

```
function setPrimaryStable(address _primaryStable) external onlyGovernor {
    require(_primaryStable != address(0), "PrimaryStable should not be empry.");
    primaryStableAddress = _primaryStable;
}
```

References

- https://halborn.com/how-centralization-enables-smart-contract-hacks-and-scams/

OGN Buyback Functionality Included In Allocate

Severity	Level of Effort	Type
Low	Low	Deprecated Functionality

Issue Description

If trusteeAddress is non-zero, IBuyback(trusteeAddress).swap() is called. With the Buyback functionality deprecated this would attempt to transfer execution to an undetermined function on the trusteeAddress. While only the Governor can set the trusteeAddress, this code is reachable via the external allocate() function and could be triggered by any Ethereum user.

Impact

If the trustee is set, this will most likely just cause a revert, but the worst case is an unexpected external function call on the trustee's address. No impact if the trustee is not set.

Supporting Evidence

Contracts/vault/VaultCore.sol:VaultCore:_allocate:341-345

```
341        // Trigger OGN Buyback
342        address _trusteeAddress = trusteeAddress; // gas savings
343        if (_trusteeAddress != address(0)) {
344            IBuyback(trusteeAddress).swap();
345        }
```

Issue Remediation

Remove the code related to Buyback functionality in _allocate()

Stable.Fi Remediation

Team Response: Implemented recommended fix and removed the code.

redeemFee Calculation Does Not Check redeemFeeBps

Severity	Level of Effort	Vulnerability Type
Low	Low	Improper Validation

Issue Description

The conditional gating of the redeemFee calculation checks labsFeeBps and teamFeeBps but not redeemFeeBps.

Impact

If redeemFeeBps is 0 or both labFeeBps and teamFeeBps are 0, the redeemFee returned will be 0, which will later cause a revert in _distributeFees, breaking redeem functionality for all users. In the expected case where all fees are nonzero, the only impact is gas spent on unnecessary operations.

Supporting Evidence

contracts/vault/VaultCore.sol:VaultCore:_calculateRedeemOutput:539-543

```
539        // Calculate redeem fee
540        if (labsFeeBps > 0 || teamFeeBps > 0) {
541            redeemFee = _amount.mul(redeemFeeBps).div(10000);
542            _amount = _amount.sub(redeemFee);
543        }
```

Issue Remediation

Use if(redeemFeeBps > 0) in _calculateRedeemOutput and check lab and team basis point factors in _distributeFees.

Stable.Fi Remediation

```
        // Calculate redeem fee
        if (redeemFeeBps > 0) {
            redeemFee = _amount.mul(redeemFeeBps).div(10000);
            _amount = _amount.sub(redeemFee);
        }
```

References

- N/A

Failure To Sanitize Code

Severity	Level of Effort	Type
Low	Moderate	Code Completeness

Issue Description

There are outdated files and functions throughout the codebase that will add size and may potentially confuse new developers (for example, VaultCore._getAssetPrices() is never used, and it is unclear whether the previous VaultCore.allocate() function is still desired). The project currently lacks documentation of its own and the testing is largely inherited. This is an artifact of the newness of the codebase but should not be overlooked as the project matures.

Impact

Code that is deprecated, unnecessary, or unused will likely cost developers time to read and users or deployers gas to deploy and use. In the worst case, an unexpected interaction could occur; while none were observed, future changes could introduce an unwanted interaction with legacy code.

Supporting Evidence

contracts/vault/VaultCore.sol:VaultCore._getAssetPrices()

The following snippet is one example of dead code that should be removed:

```
    function _getAssetPrices()
        internal
        view
        returns (uint256[] memory assetPrices)
    {
        assetPrices = new uint256[](getAssetCount());

        IOracle oracle = IOracle(priceProvider);
        // Price from Oracle is returned with 8 decimals
        // _amount is in assetDecimals
        for (uint256 i = 0; i < allAssets.length; i++) {
            assetPrices[i] = oracle.price(allAssets[i]).scaleBy(18, 8);
        }
    }
```

Issue Remediation

Remove unnecessary code. Update documentation and tests; rigorous testing of any new or modified code is essential. We recommend unit tests, dynamic testing such as fuzzing, and static analysis whenever possible.

Stable.Fi Remediation
Team Response: Specifically named functions were removed, but the team acknowledged that the codebase was not yet thoroughly cleaned up for release.

References

- N/A

Redemption Calculation Unevenly Redeems From Strategies Earliest In Array

Severity	Level of Effort	Type
Informational	Low	Code Style

Issue Description

Redemption opportunistically withdraws from the strategyWithWeights array, withdrawing all from the first strategy if possible, and continuing until redemption output is sufficient.

Impact

Strategies early in the array will be unevenly redeemed against.

Supporting Evidence

contracts/vault/VaultCore.sol:VaultCore:_redeem:184-202

```
181        uint256 totalAmount = primaryStable.balanceOf(address(this));
182        uint8 strategyIndex = 0;
183        uint8  index = 0;
184        while((totalAmount <= (output + redeemFee)) && (strategyIndex < strategyWithWeights.length)) {
185            uint256 currentStratBal = IStrategy(strategyWithWeights[strategyIndex].strategy).checkBalance();
186            console.log("Current strategy balance:", strategyWithWeights[strategyIndex].strategy, currentStratBal);
187            if (currentStratBal > 0) {
188                if ( (currentStratBal + totalAmount) > (output + redeemFee) ) {
189                    strategiesToWithdrawFrom[index] = strategyWithWeights[strategyIndex].strategy;
190                    amountsToWithdraw[index] = currentStratBal - ((currentStratBal + totalAmount) - (output + redeemFee))
191                    totalAmount += currentStratBal - ((currentStratBal + totalAmount) - (output + redeemFee));
192                } else {
193                    strategiesToWithdrawFrom[index] = strategyWithWeights[strategyIndex].strategy;
194                    amountsToWithdraw[index] = currentStratBal;
195                    totalAmount += currentStratBal;
196                }
197                index++;
198            }
199            console.log("Total amount after:", strategyWithWeights[strategyIndex].strategy, totalAmount);
200
201            strategyIndex++;
202        }
```

Issue Remediation

Randomize the order the array is visited in or use a deliberate strategy to choose which strategy to redeem from, such as choosing the strategy with the highest

balance. Assuming at least one rebalance has occurred and the strategies have been sorted by weight, the first strategy in the array will be the one with the lowest weight.

Stable.Fi Remediation
Team Response: Keeping the design as-is.

References

- N/A

Lab And Team Fees Distributed on Both Payout and Redeem

Severity	Level of Effort	Type
Informational	Low	Code Completeness

Issue Description

Fees are collected on harvest (via VaultAdmin.payout()) and on redeem, whereas OUSD only takes fees on redemption. Additionally, the intent of the calculations in VaultCore._distributeFees is unclear; for example, they reference a static 10% which is not fully explained.

Impact

The calculation of fees may not match the intent.

Supporting Evidence

Harvester.sol:_distribute -> _distributeFees:158

VaultCore.sol:_redeem -> _distributeFees:229

Harvester.sol:

```
function _distributeFees(uint256 _amount) internal {
    require(
        _amount > 0,
        "Amount should be greater than zero"
    );
    console.log("Distributing fees: ", _amount);
    uint256 labsfees = ((_amount * labsFeeBps) / 100.0) / 100.0;
    console.log("Labs fees: " , labsfees);
    uint256 teamfees = ((_amount * teamFeeBps) / 100.0) / 100.0;
    console.log("Team fees: " , teamfees);
    IERC20(primaryStableAddress).transfer(
        labsAddress,
        labsfees
    );
    IERC20(primaryStableAddress).transfer(
        teamAddress,
        teamfees
    );
```

VaultCore.sol:

```
        primaryStable.safeTransfer(msg.sender, output);
        _distributeFees(redeemFee);
        cash.burn(msg.sender, _amount);

```

```
    function _distributeFees(uint256 _amount) internal {
        require(
            _amount > 0,
            "Amount should be greater than zero"
        );
        console.log("Distributing fee:", _amount);
        uint256 labsfees = _amount.mul(labsFeeBps*10).div(10000);  // Since, we have already made the 10% of redeem amount
        console.log("labsFeeBps:", labsFeeBps);
        console.log("Sending labs fees:", labsfees);
        uint256 teamfees =  _amount.mul(teamFeeBps*10).div(10000);
        console.log("teamFeeBps:", teamFeeBps);
        console.log("Sending team fees:", teamfees);
        IERC20(primaryStableAddress).transfer(
            labsAddress,
            labsfees
        );
        IERC20(primaryStableAddress).transfer(
            teamAddress,
            teamfees
```

Issue Remediation

Add clarifying remarks in comments and overall system specification. Consider changing the argument name in _distributeFees from _amount to _redeemFee if the intent is to claim a percentage of the redeem fee.

Stable.Fi Remediation

Team Response: Changed so team fees are distributed only on profits during payout (Harvester.sol unchanged); the redemption in VaultCore.sol is changed so only the vault fee will be charged.

```

        primaryStable.safeTransfer(msg.sender, output);

        cash.burn(msg.sender, _amount);

```

References

- N/A

_calculateRedeemOutput() Contains Unnecessary Calculations

Severity	Level of Effort	Type
Informational	Low	Resource Conservation

Issue Description

The ratio variable is the result of redundant multiplication and divisions and the math surrounding definition and use of this variable could be simplified. Since much of the finance logic is inherited, some additional complexity from OUSD's multiple currencies is redundant when dealing with only a single primary stablecoin.

Impact

The math appears correct in this case, but additional complexity hurts readability and could lead to mistakes; otherwise, the extra computation just costs gas. The function _calculateRedeemOutput is an example of where math related to the primary stablecoin could be simplified, but there are likely other places in the codebase where similar simplification could reduce code complexity and gas costs.

Supporting Evidence

contracts/vault/VaultCore.sol:VaultCore:_calculateRedeemOutput

```
        IOracle oracle = IOracle(priceProvider);
        uint256 primaryStablePrice = oracle.price(primaryStableAddress).scaleBy(18, 8);
        uint256 primaryStableBalance = _checkBalance();
        uint256 primaryStableDecimals = Helpers.getDecimals(primaryStableAddress);
        uint256 totalBalance = 0;
        uint256 redeemFee = 0;
        // Calculate redeem fee
        if (labsFeeBps > 0 || teamFeeBps > 0) {
            redeemFee = _amount'.mul(redeemFeeBps).div(10000);
            _amount' = _amount'.sub(redeemFee);
        }

        totalBalance = totalBalance.add(primaryStableBalance.scaleBy(18, primaryStableDecimals));

        // Never give out more than one
        // stablecoin per dollar of CASH
        if (primaryStablePrice < 1e18) {
            primaryStablePrice = 1e18;
        }

        // Calculate totalOutputRatio
        uint256 ratio = primaryStableBalance
            .scaleBy(18,primaryStableDecimals)
            .mul(primaryStablePrice)
            .div(totalBalance);

        // Calculate final outputs
        uint256 factor = _amount'.divPrecisely(ratio);
        return (primaryStableBalance.mul(factor).div(totalBalance), totalBalance, redeemFee.div(10**(18 - primaryStableDecimals)))
```

Issue Remediation

Revisit and simplify calculations related to primary stablecoin wherever possible.

Stable.Fi Remediation
Team Response: Acknowledged.

References

- N/A

Function justMint should be removed from VaultCore

Severity	Level of Effort	Type
Informational	Low	Active Debug Code

Issue Description

This appears to be a testing function that was included as a public function in the core functionality. While the risk of including the function appears low, there is no apparent reason for an externally-accessible alternative to mint.

Impact

While assets deposited via justMint could be allocated later, having unallocated funds for any period of time would be missing potential profit. If there is more than one approved token and the user calls justMint, another user would have to trigger allocate to get those assets allocated to a strategy. In the case where an asset is approved but a default allocation strategy is not yet set (which are onlyGovernor and onlyStategistOrGovernor functions, respectively) allocate will revert. This means that if a user does justMint with such an approved asset, those assets will not be able to be allocated until a default strategy is set.

Supporting Evidence

VaultCore.sol:66-72

```
    function justMint(
        address _asset,
        uint256 _amount,
        uint256 _minimumCASHAmount
    ) external whenNotCapitalPaused nonReentrant {
        _mint(_asset, _amount, _minimumCASHAmount);
    }
```

Issue Remediation

Remove the justMint function.

Stable.Fi Remediation
Team Response: justMint is a test function, will be removed for release.

VaultAdmin._balance() Shadows A Storage Variable Name

Severity	Level of Effort	Type
Informational	Low	Code Style

Issue Description

The VaultAdmin._balance() function contains a local variable named strategies which shadows a storage variable from VaultStorage.sol:62

Impact

This fails to follow best practices, hurting readability and potentially could cause a problem in the future.

Supporting Evidence

contracts/vault/VaultAdmin.sol:VaultAdmin.sol:_balance

```
    function _balance() internal {
        IERC20 asset = IERC20(primaryStableAddress);
        StrategyWithWeight[] memory strategies = getAllStrategyWithWeights();

        // 1. calc total USDC equivalent
        uint256 totalAssetInStrat = 0;
        uint256 totalWeight = 0;
        for (uint8 i; i < strategies.length; i++) {
            if (!strategies[i].enabled) {// Skip if strategy is not enabled
                continue;
            }

            // UnstakeFull from Strategies with targetWeight == 0
            if(strategies[i].targetWeight == 0){
                IStrategy(strategies[i].strategy).withdrawAll();
            }else {
                console.log("Balance in startegy: ",IStrategy(strategies[i].strategy).checkBalance())
                totalAssetInStrat += IStrategy(strategies[i].strategy).checkBalance();
                totalWeight += strategies[i].targetWeight;
            }

        }
```

Issue Remediation

This fails to follow best practices, hurting readability and potentially could cause a problem in the future

Stable.Fi Remediation

Team Response: Refactored to use variable name stratsWithWeights instead.

```
    function _balance() internal {
        IERC20 asset = IERC20(primaryStableAddress);
        StrategyWithWeight[] memory stratsWithWeights = getAllStrategyWithWeights();
        require(stratsWithWeights.length > 0, "Weights not set");
        require(primaryStableAddress != address(0), "PS not set");

        // 1. calc total USDC equivalent
        uint256 totalAssetInStrat = 0;
        uint256 totalWeight = 0;
        for (uint8 i; i < stratsWithWeights.length; i++) {
            if (!stratsWithWeights[i].enabled) {// Skip if strategy is not enabled
                continue;
            }

            // UnstakeFull from stratsWithWeights with targetWeight == 0
            if(stratsWithWeights[i].targetWeight == 0){
                IStrategy(stratsWithWeights[i].strategy).withdrawAll();
            }else {
                console.log("Balance in startegy: ",IStrategy(stratsWithWeights[i].strategy).checkBalance());
                totalAssetInStrat += IStrategy(stratsWithWeights[i].strategy).checkBalance();
                totalWeight += stratsWithWeights[i].targetWeight;
            }

        }
```

References

- N/A