LEDGEROPS

Stabl.Fi <mark>Rebase</mark>

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 (<u>https://github.com/StablFi/stabl-contracts-audit</u>).

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.

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

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



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
VaultAdminbalance() Shadows a Storage Variable Name	Informational	Remediated
Function justMint Should be Removed from VaultCore	Informational	Remediated



CONTENTS

Stabl.Fi Rebase0
Smart Contract Audit0
EXECUTIVE SUMMARY 1
CONTENTS
METHODOLOGY
SUMMARY OF FINDINGS
harvestAndDistribute onlyGovernor Modifier Breaks Payout Functionality7
Potential Economic Attack Vulnerabilities9
Payout function Does Not Pay msg.sender15
Strategist Role Poses Potential Centralization Risk
OGN Buyback Functionality Included In Allocate20
redeemFee Calculation Does Not Check redeemFeeBps
Failure To Sanitize Code
Redemption Calculation Unevenly Redeems From Strategies Earliest In Array 25
Lab And Team Fees Distributed on Both Payout and Redeem
_calculateRedeemOutput() Contains Unnecessary Calculations
Function justMint should be removed from VaultCore
VaultAdminbalance() Shadows A Storage Variable Name



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	Туре	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	VaultAdminbalance() 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	Туре	
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.

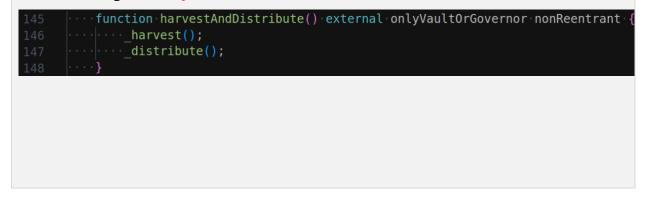
Supporting Evidence

Location: contracts/harvest/Harvester.sol:Harvestor.sol:144		
144 145 146 147	<pre>function harvestAndDistribute() external onlyGovernor nonReentrant { harvest(); distribute(); }</pre>	

Stable.Fi Remediation



Modifier changed to onlyVaultOrGovernor



References

■ <u>N/A</u>



Potential Economic Attack Vulnerabilities

Severity	Level of Effort	Туре
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-postmortem-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

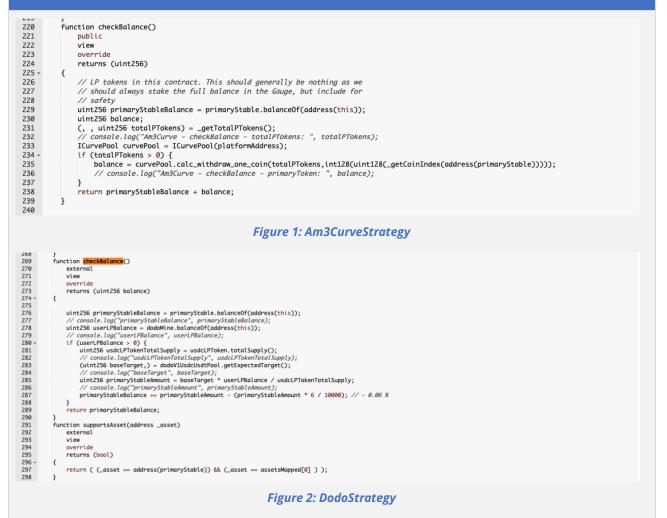
Vault	VaultCorecheckBalance and checkBalance of all strategies		
340	<pre>functioncheckBalance()</pre>		
341	····internal		
342	····view		
343	····virtual		
344	••••• returns (uint256 balance)		
345	•••••{		
346	<pre>IERC20 asset = IERC20(primaryStableAddress);</pre>		
347	<pre>balancet = asset.balanceOf(address(this));</pre>		
348			
349	<pre>for (uint256 i = 0; i < allStrategies.length; i++) {</pre>		
350	<pre>IStrategy strategy = IStrategy(allStrategies[i]);</pre>		
351	<pre>console.log("Checking Balance of ", allStrategies[i]);</pre>		
352	<pre>balancet = balancet.add(strategy.checkBalance());</pre>		
353	· · · · · · · · · · · · · · · · · · ·		
354	\cdots		

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





133 134 135 function che external view ce() override 136 137 returns (uint256) 138 { uint256 primaryStableBalance = primaryStable.balanceOf(address(this)); uint256 token0Balance; uint256 lpTokenBalance = meshSwapToken0.balanceOf(address(this)); // console.log("lpTokenBalance:", lpTokenBalance); 139 140 141 142 143 // TODO: MoreClean workground for handling non-six decimal token0
// Fix for handling token0 with 18 decimals
if (IERC20Metadata(address(token0)).decimals()) {
 IpTokenBalance = lpTokenBalance.div(10 ** (IERC20Metadata(address(token0)).decimals()) {
 IpTokenBalance = lpTokenBalance.div(10 ** (IERC20Metadata(address(token0)).decimals() - IERC20Metadata(address(primaryStable)).decimals()); // e12 = e18 - e6 144 145 146 147 148 149 150 sole.log("lpTokenBalance:", lpTokenBalance); // console.log("lprokenmaumer., specific console.log("lprokenmaumer., specific console.log("lprokenmaumer.); specific console.log("exchangeRateStored = meshSwapToken0.exchangeRateStored); // console.log("exchangeRateStored.", exchangeRateStored); token0Balance = exchangeRateStored.mul(lpTokenBalance).div(le18); 151 152 152 153 154 155 156 157 -158 -159 160 balancerVault, poolId, IVault.SwapKind.GIVEN_IN, token0, primaryStable, token0Balance.scaleBy(IERC20Metadata(address(token0)).decimals() , 6) 161 162 163 164 165 166); // console.log("Token0 swap - primaryStableBalanceFromToken0 ", primaryStableBalanceFromToken0); 167 168 169 -170 171 172 } else { primaryStableBalanceFromToken0 += token0Balance; return primaryStableBalanceFromToken0 + primaryStableBalance; 173 Figure 3: DystopiaStrategy 294 295 296 297 function checkBalance() external view override

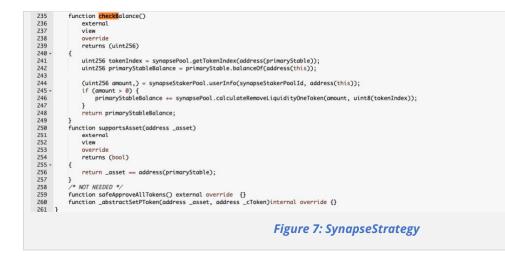


LEDGEROPS

Figure 4: MeshSwapStrategy

341	function check8alance()
342 343	external view
344 345	override
346 -	returns (uint256) {
347 348	return _totalValue(false); }
349	
350 - 351	<pre>function _collectRewards() internal { // console.log("Starting collection of rewards");</pre>
352 353	<pre>// claim rewards meshSwapPair.claimReward();</pre>
354	<pre>// console.log("claimStakingRewards called");</pre>
355 356	// sell rewards uint256 totalUsdc;
357	<pre>uint256 meshBalance = meshToken.balanceOf(address(this));</pre>
358 - 359	<pre>if (meshBalance > 10 ** 13) { uint256 meshUsdc = _swapExactTokensForTokens(</pre>
360 361	address(meshToken), address(primaryStable),
362	meshBalance,
363 364	address(this));
365 366 -	<pre>totalUsdc += meshUsdc; } else {</pre>
367	<pre>// console.log("Not enough mesh tokens to sell");</pre>
368 369	} uint256 balance = primaryStable.balanceOf(address(this));
370 - 371	<pre>if (balance > 0) { emit RewardTokenCollected(</pre>
372	harvesterAddress,
373 374	address(primaryStable), balance
375 376); primaryStable.transfer(harvesterAddress, balance);
377	
378	
	Figure 5: MeshSwapStrategyDual
260	function checkBalance()
261 262	external view
263 264	override returns (uint256)
265 -	(
266 267	<pre>uint256 token88alance = token0.balance0f(address(this)); uint256 token18alance = token1.balance0f(address(this));</pre>
268 269	<pre>uint256 lpTokenBalance = quickSwapPairStaker.balanceOf(address(this));</pre>
270 -	if (lpTokenBalance > 0) {
271 272	uint256 totallpBalance = quickSwapPair.totalSupply(); (uint256 reserve0, uint256 reserve1,) = quickSwapPair.getReserves();
273 274	<pre>token0Balance += reserve0 * lpTokenBalance / totallpBalance; token1Balance += reserve1 * lpTokenBalance / totallpBalance;</pre>
275	<pre>> constant and > = reserver = constant and > c</pre>
276 277	uint256 primaryStableBalanceFromToken0;
278 - 279 -	<pre>if ((address(token0) != address(primaryStable))) { if (token08alance > 0) {</pre>
280	primaryStableBalanceFromToken0 = onSwap(
281 282	balancerVault, poolId.
283 284	IVuult.SwapKind.GIVEN_IN,
285	token0, primaryStable,
286 287	token08alance);
288 289	<pre>// console.log("Token0 swap - primaryStableBalanceFromToken0 ", primaryStableBalanceFromToken0); }</pre>
290 -	} else {
291 292	<pre>primaryStableBalanceFromToken0 += token0Balance; }</pre>
293 294	uint256 primaryStableBalanceFromToken1;
295 -	<pre>if ((address(token1) != address(primaryStable))) {</pre>
296 - 297	if (token18alance > 0) { primaryStableBalanceFromToken1 = onSwap(
298 299	balancerVault,
300	poolId, IVault.SwapKind.GIVEN_IN,
301 302	token1, primaryStable,
303	token1Balance
304 305); // console.log("Token1 swap - primaryStableBalanceFromToken1 ", primaryStableBalanceFromToken1);
306 307 -	} }else {
308	<pre>primaryStableBalanceFromToken1 += token1Balance;</pre>
309 310	} // console.log("primaryStableBalanceFromToken0: ", primaryStableBalanceFromToken0);
311 312	<pre>// console.log("primaryStableBalanceFromToken1: ", primaryStableBalanceFromToken1); return primaryStableBalanceFromToken0 + primaryStableBalanceFromToken1;</pre>
313	<pre>return primarystablebalancerromiokeno + primarystablebalancerromiokeni, }</pre>
	Figure 6: QuickSwapStrategy





References

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



Payout function Does Not Pay msg.sender

Severity	Level of Effort	Туре
Medium	Low	Code Completeness

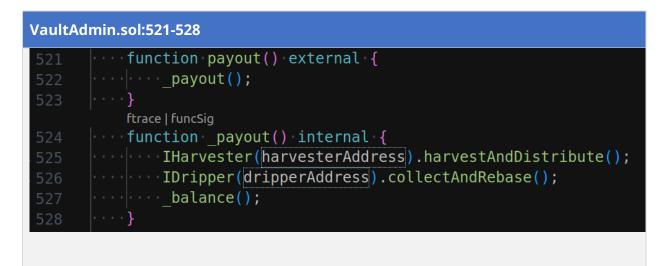
Issue Description

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



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/origindollar/blob/f0a2ce23663d32a8734d8f90b8f903ff66402961/contracts/contrac ts/harvest/Harvester.sol#L259



Strategist Role Poses Potential Centralization Risk

Severity	Level of Effort	Туре
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

415 416 417 418 419 420 421 422 423 424	<pre>function setStrategyWithWeights(StrategyWithWeight[] calldata _strategyWithWeights:) external onlyGovernorOrStrategis uint256 totalTarget == 0; for (uint8 i == 0; i <strategywithweights:).length; i++)="" {<br=""></strategywithweights:).length;></pre>
425 426 427 428 429 479 479 480	<pre></pre>
485 486 487 488	<pre>with types to the type to th</pre>
472 473 474	<pre> www.function-setPrimaryStable(addressprimaryStable*).external.onlyGovernorOrStrategis* www.primaryStableAddress.ex_primaryStable*; www.setPrimaryStable** </pre>

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.

405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420	<pre>function.setStrategyWithWeights(StrategyWithWeight[].calldatastrategyWithWeights:).external.onlyGovernor[]uint256.totalTarget:=:0;StrategyWithWeight.memory.strategyWithWeights:.length;.i++).{StrategyWithWeight.memory.strategyWithWeight.strategy].isSupported,."Strategy.should.be.supported.by.the.Vault")require(StrategyWithWeight.strategy.isSupported,."Strategy");require(strategyWithWeight.minWeight.<=:strategyWithWeight.targetWeight,);</pre>
496 497 498 499 500 501	<pre>function.setQuickDepositStrategies(address[].calldataquickDepositStrategies:).external-onlyGovernor.{for.(uint8:i==0;:i.<quickdepositstrategies:length;.i++).{require(strategies[_quickdepositstrategies:[i]].issupported,."strategy.should.be.supported.by.the.vault")}quickdepositstrategies:=quickdepositstrategies:;}< pre=""></quickdepositstrategies:length;.i++).{require(strategies[_quickdepositstrategies:[i]].issupported,."strategy.should.be.supported.by.the.vault")}quickdepositstrategies:=quickdepositstrategies:;}<></pre>
510 511 512 513 514 515	<pre>function.setSwapper(addressbalancerVault:, bytes32balancerPoolId:).external.onlyGovernor.{</pre>
484 485 486 487	<pre>function setPrimaryStable(address _primaryStable:) external onlyGovernor { require(_primaryStable: != address(0), "PrimaryStable should not be empry."); primaryStableAddress = _primaryStable:; }</pre>

References

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



OGN Buyback Functionality Included In Allocate

Severity	Level of Effort	Туре
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

Contra	Contracts/vault/VaultCore.sol:VaultCore:_allocate:341-345		
341	····//·Trigger·OGN·Buyback		
342	<pre>addresstrusteeAddress = trusteeAddress; // gas savings</pre>		
343	<pre>if (_trusteeAddress != address(0)) {</pre>		
344	<pre>IBuyback(trusteeAddress).swap();</pre>		
345	$ \cdots $		

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 <u>distributeFees</u>, 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.	Stable.Fi Remediation		
418	····//·Calculate·redeem·fee		
419	<pre>if (redeemFeeBps > 0) {</pre>		
420	<pre>redeemFee = _amountt.mul(redeemFeeBps).div(10000);</pre>		
421			
422	\cdots · · · · }		

References

■ <u>N/A</u>



Failure To Sanitize Code

Severity	Level of Effort	Туре
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:VaultCoregetAssetPrices()		
The following snippet is one example of dead code that should be removed:		
568 ····functiongetAssetPrices()		
569 ·····internal		
570 ····view		
571 ···· returns (uint256[] · memory · assetPrices +)		
572{		
573 ·····assetPrices ·=·new·uint256[](getAssetCount());		
574		
575 ···· IOracle oracle = IOracle(priceProvider);		
576 ····/·Price·from·Oracle·is·returned·with·8·decimals		
577 ····/·_amount·is·in·assetDecimals		
578 <pre>578 for (uint256 i = 0; i < allAssets.length; i++) {</pre>		
579 ••••••••••••••••••••••••••••••••••••		
580}		
581}		



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

■ <u>N/A</u>



Redemption Calculation Unevenly Redeems From Strategies Earliest In Array

Severity	Level of Effort	Туре
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

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

■ <u>N/A</u>



Lab And Team Fees Distributed on Both Payout and Redeem

Severity	Level of Effort	Туре
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

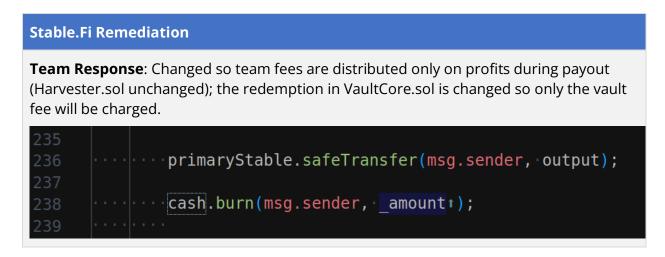
Harvester.sol:_distribute -> _distributeFees:158		
VaultCore.sol:_redeem -> _distributeFees:229		
Harvester.sol:		
<pre>158</pre>		
159 ···· require(
160 \cdots \cdots \cdots amount $t > 0$,		
161 ···· Amount should be greater than zero"		
162);		
<pre>163</pre>		
164uint256 labsfees = ((_amount * * labsFeeBps) / 100.0) / 100.0;		
165 console.log("Labs fees: ", labsfees);		
166uint256 teamfees = ((_amount * teamFeeBps) / 100.0) / 100.0;		
<pre>167</pre>		
168 IERC20(primaryStableAddress).transfer(
169 ····labsAddress,		
170 ·····labsfees		
171 ();		
172 IERC20(primaryStableAddress).transfer(
173 ···· teamAddress,		
174 ···· ····teamfees		
175 ·····);		



VaultCore.sol:
215
<pre>216 primaryStable.safeTransfer(msg.sender, output); 217 distributeFees(redeemFee);</pre>
<pre>218 cash.burn(msg.sender, amountt);</pre>
219
<pre>229 ····function·_distributeFees(uint256·_amount:)·internal·{ 230 ·····require(231 ·····amount:·>·0, 232 ·····amount:·>·0, 233 ·····amount:·>·0, 233 ·····amount:·>·0, 234 ·····amount:·>·0, 235 ·····amount:·>·0, 236 ·····amount:·>·0, 237 ·····amount:·>·0, 238 ·····amount:·>·0, 239 ······amount:·>·0, 239 ·····amount:·>·0, 239 ······amount:·>·0, 239 ······amount:·>·0, 239 ·····amount:·>·0, 239 ······amount:·>·0, 239 ·······amount:·>·0, 239 ····································</pre>
<pre>234</pre>
<pre>237 ·····console.log("Sending labs fees:", labsfees); 238 ······uint256 teamfees =amount .mul(teamFeeBps*10).div(10000); · 239 ······console.log("teamFeeBps:", teamFeeBps);</pre>
240 ······console.log("Sending team fees:", teamfees); 241 ······IERC20(primaryStableAddress).transfer(242 ············labsAddress,
242 ···································
245 ······IERC20(primaryStableAddress).transfer(246 ·········teamAddress,
247 ····· 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.



🔘 LEDGEROPS

References

■ <u>N/A</u>

_calculateRedeemOutput() Contains Unnecessary Calculations

Severity	Level of Effort	Туре
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





Issue Remediation

Revisit and simplify calculations related to primary stablecoin wherever possible.

Stable.Fi Remediation

Team Response: Acknowledged.

References

■ <u>N/A</u>



Function justMint should be removed from VaultCore

Severity	Level of Effort	Туре
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

Vault	VaultCore.sol:66-72		
66	••••function justMint(
67	•••••address _asset + ,		
68	•••••uint256 _amount*,		
69	••••••uint256•_minimumCASHAmount		
70	<pre>) external whenNotCapitalPaused nonReentrant {</pre>		
71	<pre>mint(_asset*, _amount*, _minimumCASHAmount*);</pre>		
72	\cdots		



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	Туре
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

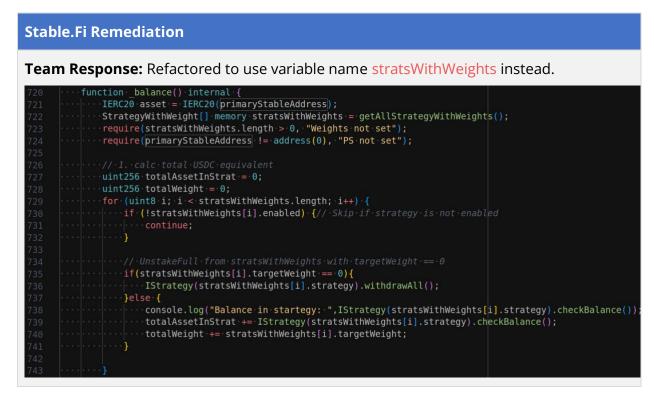
contracts/vault/VaultAdmin.sol:VaultAdmin.sol:_balance



536	<pre>function balance() internal {</pre>
537	<pre>IERC20 asset = IERC20(primaryStableAddress);</pre>
538	<pre>StrategyWithWeight[] memory strategies = getAllStrategyWithWeights();</pre>
539	
540	····/·1.·calc·total·USDC·equivalent
541	·····uint256·totalAssetInStrat ==0;
542	·····uint256·totalWeight·=·0;
543	····· for·(uint8·i; i < strategies.length; i++) {
544	······························if·(! <mark>strategies</mark> [i].enabled)·{//·Skip·if·strategy·is·not·enabled
545	······continue;
546	·····}
547	
548	······································
549	<pre>if(strategies[i].targetWeight == 0){</pre>
550	<pre>IStrategy(strategies[i].strategy).withdrawAll();</pre>
551	····}else-{
552	<pre>console.log("Balance in startegy: ",IStrategy(strategies[i].strategy).checkBalance())</pre>
553	<pre>totalAssetInStrat += IStrategy(strategies[i].strategy).checkBalance();</pre>
554	<pre>totalWeight += strategies[i].targetWeight;</pre>
555	
556	
557	····}

Issue Remediation

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



References

■ <u>N/A</u>

EDGEROPS