Five Pillar

FINAL REPORT

July ‘2025

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Five Pillar – Audit Report
Website	
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/fivepillarstoken/InvestmentManager/blob/652bff9fdc72b37bc3b976e236d0b8fd5928d40f/InvestmentManager.sol https://github.com/fivepillarstoken/5PT-Token/blob/2478b01e8d230fd4ce6756555eecb4e67d4ff74f/FivePillarsToken.sol
Resolution 1	https://github.com/fivepillarstoken/InvestmentManager/blob/986104eb80159e7c0dcaa81953260a1a48c05183/InvestmentManager.sol
Resolution 2	https://github.com/fivepillarstoken/InvestmentManager/blob/789ecd271827bbbf8859a0140c469088144ecba9/InvestmentManager.sol
Resolution 3	https://github.com/fivepillarstoken/InvestmentManager/blob/59b5dd477629c3fa7fdb3e71abdaade1dbed2702/InvestmentManager.sol

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High						
Medium	10	8		2		
Low	5	5				
Informational	7			7		
Governance						
Total	22	13		9		

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

2. Detection

Audit Findings Summary

During our audit, we identified two fundamental conceptual mistakes that led to multiple downstream issues:

1. **Redistributions Not Treated as Deposits**
   Redistributed amounts are not handled in the same way as direct deposits, leading to inconsistencies.

2. **Deposits Use Outdated Amounts**
   The system uses non-updated deposit values, which introduces additional problems.

Key Questions for Design Re-evaluation

When considering fixes, we recommend reflecting on the broader system context. Important questions to address include:

- **Redeposit Behavior**
  Should redeposits be treated identically to new deposits?

- **Referral-only Balances**
  Should it be possible for a user to gain a balance exclusively through referral rewards (fees and redeposits), without ever depositing directly?

- **Fee Inclusion in Pool Calculations**
  Should collected fees be factored into pool reward computations?

- **Treasury Management**
  Currently, the treasury cannot be updated. Is this intentional?

- **Minimum Deposit Threshold**
  Should there be a minimum deposit requirement (e.g., 1 token) for all deposits?

- **Fairness for Early Depositors**
  As it stands, early depositors gain no advantage over those depositing later in a round. Is this correct?

- **Fee Exclusion in Reward Calculation**
  The getAccumulatedRewards() function does not account for fees. Is this correct?

- **Investor Pool Membership Check**
  Introduce a simplified mechanism to determine if an investor is part of a pool — for example, by adjusting getInvestorPoolRewardPerTokenPaid().

- **Pool Tier Integrity**
  All pools can currently be configured with identical requirements, potentially undermining the intended tier structure. Is this intentional?

Resolution Round

As per our contract, intrusive changes (even if recommended), out-of-scope changes and refactoring may require a re-audit of the whole codebase if code-changes expose a certain risk. Therefore, we highly recommend the developer to only introduce an absolute minimum of code-changes, even if that means acknowledging multiple issues.

This reflects our goal to reach a deployment-ready codebase after the resolution round.

FivePillarToken

The FivePillarToken is an ERC20 token contract with an initial supply of 100 billion which is minted to the owner during construction. It implements Openzeppelin's Ownable2Step for two-step ownership transfer mechanism. Only the InvestmentManager address can mint and burn these tokens. This token works as the core token (deposit token and reward token) in the InvestmentManager contract.

Core Invariants:

INV 1: Only the InvestmentManager can mint and burn tokens.
INV 2: Only the owner can assign a new investment manager.

Privileged Functions

- setInvestmentManager
- mint
- burnFrom

InvestmentManager

The InvestmentManager is designed to manage investments, rewards, and pool participation within the Five Pillars ecosystem. It integrates with OpenZeppelin's SafeERC20 for secure token handling and Ownable2Step for two-step ownership transfer, and interacts with the Five Pillars token and PancakeSwap router via defined interfaces. The contract implements a multi-tier reward system comprising daily rewards based on investment amounts, referral rewards for direct and indirect referrals, and pool rewards tied to investment tiers and criteria.

Key features include: a deposit system where users can invest tokens after a specified start timestamp, with a minimum deposit of 1 token for first-time deposits and a 4-hour delay between deposits. Deposits incur a fee, which is collected and swapped to ETH via PancakeSwap for distribution to two treasury addresses.

The reward system calculates daily rewards at 0.3% of an investor's total deposit per round (24 hours), referral rewards at 2.5% of direct referrals' deposits (plus 0.675% of downline deposits for pool 2 participants), and pool rewards based on nine predefined pools with varying criteria. Pools 0–6 require specific personal investment, direct referral counts, and referral deposit thresholds, while pools 7–8 are whitelist-only, managed by the contract owner. Pool rewards are distributed proportionally based on each pool's share (in basis points) and participant count.

Investors can claim accumulated rewards, subject to a claim fee, with 50% of claimed rewards redistributed to the ecosystem, updating referral and pool rewards. The contract enforces a 30-hour delay between pool criteria updates, which are processed in batches to respect gas limits, ensuring pool hierarchy is maintained (criteria must not decrease by more than half).

Core Invariants:

INV 1: Deposits are only accepted after startTimestamp and with a delay.
INV 2: Deposit and claim fees are in their specified ranges.
INV 3: First deposits and reward claims must be ≥ 1 token (10^18 wei).
INV 4: An investor's referer, once set, is immutable
INV 5: Pool criteria updates for pools 0–6 maintain hierarchy: each subsequent pool has equal or higher personalInvestRequired, directRefsRequired, and totalDirectInvestRequired.
INV 6: Pools 7 and 8 are accessible only to whitelisted investors
INV 7: A pool's isActive status is true if and only if participantsCount > 0.
INV 8: totalDepositAmount accurately reflects the sum of all investors' totalDeposit after deposits and reward redistributions.

INV 9: isUpdateCriteriaActive prevents deposits and concurrent criteria updates until all investors are processed.

Privileged Functions

- setDepositFee
- setClaimFee
- setWhitelist
- setPoolCriteria

Issue_01	deposit and claimReward incorrectly use the old deposit instead of the new one when adding investors to the pool
Severity	**Informational**
Description	The investment contract includes multiple pools, each with different eligibility requirements for receiving rewards. However, when a user deposits enough funds that meets a pool's threshold, they may still not be added to the pool and begin accruing rewards immediately. This happens because the _checkAndAddInvestorToPool() function only considers the user's **existing totalDeposit**, not the **current amount being deposited**, when checking eligibility. As a result, the user is only added to the correct pool on their **next deposit**, or when they are used as a **referrer** by another depositor. This behavior seems unintended. The same issue appears in claimRewards. When half of the rewards are redistributed to the protocol, that redistributed amount is not considered when evaluating eligibility for entering a pool. Furthermore, because the depositor does not enter the pool, the purpose of curReward is undermined, as the deposited amount does not contribute to curReward when it realistically should. Notably, referrers are handled correctly. Their directRefsDeposit and directRefsCount are updated **before** checking their eligibility, ensuring that they are added to the pool at the right time. The depositor should be handled in a similar way.
Recommendations	Consider updating the depositor's totalDeposit amount before invoking _updatePoolRewards to make sure they are correctly added to their respective pool once criteria is met. Another option would be to pass the amount as a parameter to the *_checkAndAddInvestorToPool()* function. Then, inside that function, one can add it to the user's totalDeposit (totalDeposit + amount) to include it.

Comments / Resolution	Acknowledged. The client indicated this is a design issue. It is downgraded from high to informational severity.

Issue_02	A user can inflate their indirect referrals by creating a circular referral chain with two or more accounts.
Severity	**Medium**

Description	When depositing into the InvestmentManager contract, a user can set any account as a referrer. The problem is that this allows circular referral chains to be created using multiple accounts, which leads to a loop that artificially inflates indirect referral metrics.
	
	Let me walk through the process:
	
	- Bob owns two accounts: accountA and accountB.
	- First, he deposits with accountA, setting accountB as the referrer.
	- Then, he deposits with accountB, setting accountA as the referrer.
	- The problem occurs inside the _updateReferers() function. When Bob deposits with accountB, it updates the referrer (accountA), and then enters a loop that repeatedly calls _updateDownlineReferer() using accountA and accountB alternately, up to 9 times:
	
	function _updateReferers(address referer, uint256 amount, bool isFirstDeposit) internal {
	    [...]
	
	>> for (uint i = 0; i < 9; i++) {
	>> referer = accountToInvestorInfo[referer].referer;
	* if (referer == address(0)) break;*
	>> _updateDownlineReferer(referer, amount, isFirstDeposit);
	* }*
	* }*
	
	- Inside _updateDownlineReferer(), each account receives credit for the amount deposited by accountB. Since it's the first deposit for

accountB, it also inflates the indirect referrals count of each account:

```
function _updateDownlineReferer(address referer, uint256 amount,
bool isFirstDeposit) internal {
        _updateInvestorRefReward(referer);

>>      accountToInvestorInfo[referer].downlineRefsDeposit +=
amount;
>>      if (isFirstDeposit)
accountToInvestorInfo[referer].downlineRefsCount += 1;
    }
```

- Both accounts end up with more indirect referrals (1) and inflated downlineRefsDeposit(2) values.

1. As it is nine iterations the indirect referrals of each account end up:
    - *AccountA.downlineRefsCount = 4*
    - *AccountB.downlineRefsCount = 5*

    NOTE: Since downlineRefsCount only increases on first deposits, Bob could introduce a third account referring to accountB to further amplify the inflation.

2. Similarly, downlineRefsDeposit is inflated without needing new accounts. Simply depositing more with either account keeps increasing the value.

If the investor is in the third pool it can get an inflated roundReward:

```
        uint256 roundReward = investorInfo.directRefsDeposit * 2500 /
BASIS_POINTS;

        if (isInvestorInPool[investor][2]) {
            // 0.00675%
>>          roundReward += investorInfo.downlineRefsDeposit * 675 /
BASIS_POINTS;
        }
```

	*return (roundReward * endedRounds, roundReward);*
Recommendations	There are two possibilities: - Do not allow users to set a referrer that is not already part of the investment strategy. However, this might go against the protocol's purpose, as allowing referrals from outside the protocol can encourage broader participation. - Inside the for loop, keep track of all the referrers encountered. If a referrer appears more than once, break the loop to avoid entering a circular referral chain. ***It has to be noted that intrusive logical change for this algorithm will require a full re-audit and is not covered within a simple resolution round as this means the whole control-flow must be rechecked.***
Comments / Resolution	Resolved.

Issue_03	A user can inflate their indirect referrals by creating a circular referral chain with two or more accounts after first deposit.
Severity	**Medium**

Description	The protocol team decided to implement the possibility to set a referral even if the deposited amount > 0.
	This allows them to claim rewards and later deposit with a referral.
	
	This removes the constraint of setting the referral only on the first deposit, skipping important security checks, enabling the above mentioned attack again.
	Now the user can first deposit without a referral and on the second deposit activate the circular referral chain.
	
	This is due to treating referral settings, and first deposit independently, **see the previous implementation**
	
	`if (referer != address(0)) {`
	    `if (investor.totalDeposit > 0) revert RefererAlreadySetted();`
	    `if (investorAddress == referer) revert InvalidReferer();`
	    `investor.referer = referer;`
	`}`
	`bool isFirstDeposit = investor.totalDeposit == 0;`
	`if (isFirstDeposit) {`
	    `_checkDepositOrClaimAmount(amount);`
	    `investor.referer = referer;`
	    `_checkRefererCirculation(referer);`
	
	Now the new implementation:
	
	`if (referer != address(0)) {`
	    `if (investor.referer != address(0)) revert RefererAlreadySetted();`
	    `if (investorAddress == referer) revert InvalidReferer();`
	    `investor.referer = referer;`
	`}`
	`bool isFirstDeposit = investor.totalDeposit == 0;`
	`if (isFirstDeposit) {`
	    `_checkDepositOrClaimAmount(amount);`
	    `investor.referer = referer;`
	    `_checkRefererCirculation(referer);`

	_investors.push(investorAddress);
	if (isWhitelisted[investorAddress][7] || isWhitelisted[investorAddress][8]) onlyWhitelistedInvestorsCount -= 1;
	}
	
	As we can see, the _checkRefererCirculation() check is only done when the user deposits for the first time. However not when the referral above is set. Allowing to earn more downstream referral rewards then expected.

Recommendations	While a potential fix is to move the _checkRefererCirculation() check inside the above if case, right **below** where investor.referer = referer; is set. It furthermore requires the removal of both checks inside the first deposit check. It is important to always check for referrer circulation after the referrer is set. Important note: The validation of this issue requires a complete new validation round where all control-flows need to be validated, this requires additional charge and is not part of the initial audit and resolution round. The likelihood is high that a new issue is being introduced by this fix. We recommend the client to either acknowledge this issue or implement a fix and schedule a separate audit for this fix.
Comments / Resolution	Resolution 3: The check has been reverted and investor.totalDeposit > 0 is now used as check. This issue is resolved.

Issue_04	User can bypass criteria updates by not being included in the _investors array
Severity	Medium
Description	When a user makes a deposit, they can set any address as a referrer—even one that hasn't interacted with the contract yet. If enough time passes, the referrer can later call claimRewards() to collect referral rewards, even though they never made a deposit themselves. Thus, they don't receive pool/daily rewards—only referral rewards. Here's the problem, if the referrer's first interaction with the protocol is through claimRewards() (not deposit()), they are never added to the _investors array. The _investors array is used when an admin updates pool criteria. The system loops through _investors to validate eligibility. Since the referrer was never added to the array, he is skipped during these checks, allowing them to remain in pools they no longer qualify for. An attacker can exploit this by creating a minimal account and setting as a referrer his main account. Since the referrer is not added to the _investors array, he can later claim rewards and begin depositing without ever being subjected to pool criteria updates. It happens the same thing to the whitelisted users of pools 8 and 9, when a whitelisted is set, after some time his first interaction can be through claimRewards() and will happen the same. Additionally the whitelisted status will not change, this does not have any major impact, but leads to an incorrect whitelisted amount. Additionally the whitelisted status will not change, this does not have any major impact, but leads to an incorrect whitelisted amount.

Recommendations	In claimRewards() consider checking whether the investor has a investor.totalDeposit == 0. If yes, consider adding it into the _investors array.
Comments / Resolution	Resolved.

Issue_05	Referrer counts not updated when referrer is set after first deposit
Severity	**Medium**

Description	A user can set a referrer at any time, including after their first deposit. This flexibility has been introduced as fix to another issue, which makes it a total of at least two issues which were introduced by this fix. Since directRefsCount and downlineRefsCount are only incremented when isFirstDeposit is true, setting a referrer after the first deposit results in these counters not being updated correctly.
	
	function _updateDownlineReferer(address referer, uint256 amount, bool isFirstDeposit) internal {
	    _updateInvestorRefReward(referer);
	
	    accountToInvestorInfo[referer].downlineRefsDeposit += amount;
	>> *if (isFirstDeposit) accountToInvestorInfo[referer].downlineRefsCount += 1;*
	  }
	
	  function _updateReferers(address referer, uint256 amount, bool isFirstDeposit) internal {
	    _updateInvestorRefReward(referer);
	
	    accountToInvestorInfo[referer].directRefsDeposit += amount;
	>> *if (isFirstDeposit) accountToInvestorInfo[referer].directRefsCount += 1;*
	
	    for (uint i = 0; i < 9; i++) {
	      referer = accountToInvestorInfo[referer].referer;
	      if (referer == address(0)) break;
	      _updateDownlineReferer(referer, amount, isFirstDeposit);
	    }

	`}` If a referrer is set after the first deposit, their referral counts will not increase, even though they should. This can prevent eligible users from qualifying for pools.
Recommendations	Consider simply reverting to the previous version, using totalDeposit as source of truth for referrer setting, while acknowledging the previous medium severity issue for the favor of fixing the two newly introduced issues.
Comments / Resolution	Resolution 3: The check has been reverted and investor.totalDeposit > 0 is now used as check. This issue is resolved.

Issue_06	Missing rewards for a newly activated pool through setPoolCriteria()
Severity	**Informational**
Description	When a pool is activated for the first time through setPoolCriteria(), its rewardPerInvestorStored is initialized to zero. As a result, newly eligible investors will not accrue any rewards for that pool until new deposits are made. Their existing totalDeposit is not accounted for in the pool's curReward, meaning they receive no rewards from a higher-tier pool they now qualify for. This leads to several inconsistencies: - Investors upgraded into the pool, often the largest depositors, will receive no rewards despite meeting the criteria. - This violates the protocol's stated intent that higher-tier pools yield greater returns. We can contrast this with the reactivation of a previously active pool, where curReward is preserved (not reset), allowing rewards to continue accruing immediately once an eligible investor reactivates the pool. This highlights the inconsistency: reactivated pools fairly resume reward accrual, while newly activated pools effectively delays their earnings.
Recommendations	When activating a pool for the first time, aggregate the totalDeposit of all investors who qualify and immediately update the pool's curReward accordingly. Since the pool was previously inactive, no rewards duplication will occur. This approach aligns with the protocol's design principle of rewarding all active deposits within eligible pools and ensures fairness. Another approach would be to update the curReward on each deposit of each pool even if inactive. Then when the pool is activated the curReward would be updated already and the

	newcomers will start accrue rewards immediately from all the deposits made in the protocol. But I don't how this can be aligned with the protocol's purposes. ***It has to be noted that intrusive logical change for this algorithm will require a full re-audit and is not covered within a simple resolution round as this means the whole control-flow must be rechecked.***
Comments / Resolution	Acknowledged. With comment: Users are expected to actively interact with the protocol. The developer provided docs that this is a business logic decision. The severity has been downgraded from medium to informational.

Issue_07	Fee swap has slippage disabled
Severity	**Medium**
Description	The `_trySendFees` function is invoked at the end of every deposit and claim in order to attempt to swap the FPT accumulated fees into WETH and send them to the treasuries. However, the PancakeRouter01 call to `swapExactTokensForETH` has both a disabled deadline and slippage set to 0, meaning that value could be stolen from the unprotected fee swap
Recommendations	Consider introducing an acceptable slippage threshold for swapping FTP to WETH for fees. **It has to be mentioned that this will require the implementation of a permission parameter as it will not be possible to introduce a slippage check without a reliable parameter input. Ideally, a simple onlyOwner function is implemented for this purpose and it is not triggered during deposits.**
Comments / Resolution	Partially fixed, the current implementation forces the owner to update the price frequently and can result in issues if price is stale. This has been fully fixed during resolution 2.

Issue_08	Remaining gas computation during setPoolCriteria is incorrect and could lead to OOG
Severity	**Medium**

Description	The setPoolCriteria function, meant to change pool criterias and update investors' eligible pools, tracks 2 values for safety, a checkCountLimit to keep track of the number of iterations and gasSpent to avoid DOS, limiting the gas usage to ⅔ of the block limit. However, gasSpent is incorrectly set as lastGasLeft - gasleft(), where lastGasLeft is set to gasleft() just above this calculation:
	`lastGasLeft = gasleft();`
	
	`gasSpent += lastGasLeft - gasleft();`
	This essentially leads to gasSpent being 0, completely avoiding the OOG protection at the end of the function.

Recommendations	Consider either fixing the gas spending calculator or removing it altogether, a manual limit on loop iterations should be sufficient to avoid OOG.
Comments / Resolution	Resolved. In the end of the loop checkCount is first increased to match the next "i", then it's checked if checkCount is larger or equal then the limit specified, in case yes the currentId is set to i+1 (next item in loop) and the function returns.

Issue_09	Pool is not activated when setting the pool criteria
Severity	**Medium**
Description	When updating the criteria for certain pools and iterating over all investors, if a new investor becomes eligible for a higher pool that is currently inactive, the pool is not activated. As a result, it will not accrue rewards in subsequent blocks until an eligible user makes a deposit, which then will trigger its activation.
Recommendations	When adding an investor to a pool via setPoolCriteria(), verify that the pool is active.
Comments / Resolution	Resolved.

Issue_10	Reward calculations vary inconsistently depending on the amount used to activate a pool
Severity	**Medium**
Description	The current system calculates curReward only for active pools, which introduces inconsistencies in reward distribution based on the specific conditions under which a pool is activated. To illustrate the issue, I will use Pool 2, which has a threshold of 1,450,000e18. The following two scenarios demonstrate how the same user may receive vastly different rewards depending on how the pool is activated: **Scenario 1 (Pool activated with minimal amount)**: - User deposits 1,449,999e18, making them eligible only for Pool 1. - Then deposits an additional 1e18, triggering Pool 2 activation. - curReward for Pool 2 is calculated using just 1e18, resulting in curReward = 0.000175e18. - Reward per investor for Pool 2 = 0.000175e18. **Scenario 2 (Pool activated with full amount):** - User deposits the entire 1,450,000e18 in one go, qualifying directly for Pool 2. - curReward is calculated using the full amount, resulting in curReward = 253.75e18. - Reward per investor for Pool 2 = 253.75e18. In both cases, the user ends up eligible for Pool 2, but due to the different activation contexts, receives vastly different rewards. This breaks consistency and creates potential edge cases where users

	are penalized for how they activate a pool, particularly problematic when higher-tier pools are involved.
Recommendations	When a pool is activated, use the full totalDeposit of the eligible users to calculate curReward, rather than just the amount that triggered the activation. This ensures that the reward calculation reflects the true stake of the investor in the system and avoids penalizing users who activate a pool with a minimal top-up amount. It aligns better with the intent of rewarding users based on their total participation rather than the specific deposit that changes their eligibility. *It has to be noted that intrusive logical change for this algorithm will require a full re-audit and is not covered within a simple resolution round as this means the whole control-flow must be rechecked. This issue can also be acknowledged as design-choice.*
Comments / Resolution	Acknowledged. With comment: Design choice, only activated amount should count towards rewards. The original severity remains because we believe this is still an issue and the severity is technically appropriate.

Issue_11	Manually setting minSwapPrice can be problematic
Severity	**Medium**
Description	To handle slippage in the swap between fivePillarsToken and WETH a new variable has been introduced i.e. minSwapPrice. For instance if the minSwapPrice is 5 and the balance of fivePillarsToken in the contract is 10 then minimum out amount should be 50 WETH. But due to volatility of tokens and market the swap price is bound to change , lets assume after a while swap price jumped to ~ 10 therefore for 10 fivePillarsToken we should get somewhere around 100 ETH and before the owner could update the minSwapPrice a user calls deposit() or claimReward() which swaps fivePillarsToken to ETH and in this swap the old minSwapPrice i.e. 5 would be used making the amountOutMin to 50 , this means an attacker can still extract value out of this trade for profit.
Recommendations	Consider following the initial recommendation and implementing a governance function for swapping.
Comments / Resolution	Resolved during resolution 2.

Issue_12	A user can be denied from setting a referrer if they claim first
Severity	**Medium**

Description	A user is eligible to set a referrer when the user's totalDeposit is 0 →
	
	 `if (referer != address(0)) {`
	`        if (investor.totalDeposit > 0) revert RefererAlreadySetted();`
	`        if (investorAddress == referer) revert InvalidReferer();`
	`        investor.referer = referer;`
	`    }`
	
	 But users can be added to pool 7 and 8 without ever making a deposit and therefore will earn rewards which can be claimed through the claimRewards() function which in turn increases the investor's totalDeposit. Since the totalDeposit is non-zero now , the investor is not allowed to set a referrer on deposits. Therefore , this user/investor is prevented from ever assigning a referrer due to totalDeposits being non zero upon claiming rewards.
	
	 This is similar to the issue *User can bypass criteria updates by not being included in the _investors array* , where the underlying problem was same i.e. non-zero totalDeposits without ever interacting with the deposit function which lets them bypass criteria updates , though the impact here is limited to the user himself.

Recommendations	If the totalDeposit has been updated via claimReward then allow the user to assign a referrer.
Comments / Resolution	Resolution 3: The change has been reverted and the issue is now considered as acknowledged

Issue_13	Rewards won't accrue if some timestamp is updated in the same block as startTime
Severity	**Low**
Description	There are three reward streams that rely on timestamps to track accrual: 1. Pool rewards: uses lastUpdatePoolRewardTimestamp. 2. Daily Rewards: uses lastDepositTimestamp or lastClaimTimestamp. 3. Referral Rewards: uses investor.updateRefRewardTimestamp. If any of these timestamps are set in the same block as startTime, no rewards will accrue for that stream until the timestamp is updated again. This happens because of the following line inside _calcCountOfRoundsSinceLastUpdate(): *if (lastUpdate <= startTime) return 0;* On the next update, since lastUpdate is equal to startTime, the function returns 0, skipping accrual entirely even if time has passed.
Recommendations	Consider properly handling this edge-case.
Comments / Resolution	Resolved.

Issue_14	Users may be eligible for a pool but fail to qualify due to unclaimed rewards.
Severity	**Informational**
Description	When evaluating whether an investor meets the criteria for a higher-tier pool (via _checkAndAddInvestorToPool() or setPoolCriteria()), the system uses totalDeposit without first claiming pending rewards. However, claiming rewards triggers auto-compounding, increasing the user's totalDeposit. Without doing this first, an investor might appear ineligible despite technically qualifying after compounding. Since users are not required to know when their eligibility will be checked, they may miss out on joining a pool they're effectively eligible for. This leads to lost potential rewards from the higher-tier pool.
Recommendations	Automatically call claimRewards() for the user before checking their eligibility for a pool. This ensures their totalDeposit reflects compounded rewards and prevents missing pool entry due to unclaimed earnings.
Comments / Resolution	Acknowledged with comment: Protocols goal is active user participation, they are required to manually claim before. This issue has been downgraded.

Issue_15	getLastRoundRewards() breaks earlier than intended
Severity	**Low**

Description	getLastRoundRewards() breaks earlier than it should in the case where a round hasn't passed since the last update for the investor, entering the last else clause:
	`else {`
	`  for (uint8 i = 0; i < pools.length; i++) {`
	`if(!isInvestorInPool[investorAddress][i]) {`
	`                break;`
	`            }`
	
	`            poolsReward += pools[i].lastReward;`
	`        }`
	`    }`
	
	We iterate over all pools and break if the investor is not in the current pool, due to the sequential assumption that not being in pool 2, means you cannot be in 3...9. However 8 and 9 are whitelist based, so breaking here skips them over, even though it is possible to have a deposit and accumulated rewards there, returning an inaccurate result.

Recommendations	Consider adding a separate loop for the 2 whitelist pools so the view function returns appropriate results
Comments / Resolution	Resolved.

Issue_16	onlyWhitelistedInvestorsCount can be inflated if the same user is whitelisted on both pools 8 and 9.
Severity	**Low**

Description	The setWhitelist() function is used to whitelist users for Pools 8 and 9. If a user is whitelisted and has not yet deposited, the protocol increases the onlyWhitelistedInvestorsCount counter:
	
	if (accountToInvestorInfo[investor].totalDeposit == 0)
	onlyWhitelistedInvestorsCount += 1;
	
	However, if the same user is whitelisted for both Pools 8 and 9, the counter is incremented twice, once for each pool, even though it's the same user. Later, when this user deposits, the counter will only be decremented once, but the investor will still be added just once to _investors array. This causes the total investor count to become inconsistent.
	
	function getTotalInvestorsCount() external view returns(uint256) {
	    return _investors.length + onlyWhitelistedInvestorsCount;
	  }

Recommendations	In setWhitelist(), before incrementing onlyWhitelistedInvestorsCount, check whether the user is already whitelisted for the other pool. Only increment the counter if the user is being newly added to either pool and has not already been counted. This guarantees accurate accounting of the total investor count. Alternatively, this issue can also be acknowledged in an effort to not further implement code changes since this issue does not have any financial impact.
Comments / Resolution	Resolved.

Issue_17	onlyWhitelistedInvestorsCount is not decremented when a whitelisted user's first interaction is via claimRewards().
Severity	**Low**
Description	When a user is whitelisted for Pools 8 and 9 but has not yet deposited, the protocol increases the onlyWhitelistedInvestorsCount with the following condition: *if (accountToInvestorInfo[investor].totalDeposit == 0) onlyWhitelistedInvestorsCount += 1;* Over time, the user begins to receive rewards from these pools. If the first interaction with the protocol is through claimRewards(), half of those rewards are added to the user's totalDeposit, effectively acting as a "first deposit". However, because this flow bypasses the usual deposit logic, onlyWhitelistedInvestorsCount is not decremented to reflect the user's new status. This causes an inconsistency: the user is now an active investor, but is still counted as "whitelisted only," and they are not added to the _investors array either.
Recommendations	Inside claimRewards check if the claimer is inside the 8 and 9 pools and his totalDeposit is zero. Then, if true, decrement onlyWhitelistedInvestorsCount and add the investor into the _investors array. **NOTE**: here you can handle the solution of the issue below, so when an investor is whitelisted for both pools and his first interaction is through claimRewards, then check as well that user is not in both pools, then if true decrement properly the whitelist counter. **Alternatively, this issue can also be acknowledged in an effort to not further implement code changes, since this issue has no financial impact.**

Comments / Resolution	Resolved.

Issue_18	Fees can be dodged on every deposit after the first one
Severity	**Informational**
Description	The deposit function checks a minimum deposit amount on the very first deposit, to make sure users are adequately invested into the system. However, subsequent deposits allow arbitrary sized values and this could be used to deposit very small amounts that dodge the fee accumulation, due to rounding.
Recommendations	The per deposit delay defeats most of the incentive, but it should be considered having a minimum deposit amount on every deposit, not just the first
Comments / Resolution	Acknowledged. This issue has been downgraded because the client expressed this is a design choice.

Issue_19	setPoolCriteria incorrectly sets the _updateCriteriaCurInvestorId to the current id instead of the next
Severity	**Low**
Description	The setPoolCriteria function uses a gas limit and an iteration cap to protect itself from OOG when going over and moving every investor between pool tiers. When we reach those caps, the function saves the index we reached for it to start again from it on the next call. However, this is done at the end of the loop when the investor was already updated and the function does _updateCriteriaCurInvestorId = i meaning we start the next loop from the same investor twice
Recommendations	Change the line _updateCriteriaCurInvestorId = i to _updateCriteriaCurInvestorId = i+1
Comments / Resolution	Resolved.

Issue_20	Improper amount used when updating pool rewards during deposit
Severity	**Informational**
Description	During a deposit, the protocol deducts a fee from the deposited amount. The remaining amount (toInvestor = amount - fee) is then used to update various components such as referral data and the user's totalDeposit. However, when updating the pool rewards, the full amount (including the fee) is passed instead: *if (investor.referer != address(0)) _updateReferers(investor.referer, toInvestor, isFirstDeposit);* *>> _updatePoolRewards(amount, investorAddress, investor.referer);* *investor.totalDeposit += toInvestor;* *totalDepositAmount += toInvestor;* This behavior differs from the claimRewards() function, where the same value (`toRedistribute`) is consistently used across all operations—updating referrals, pool rewards, and deposits: *if (investor.referer != address(0)) _updateReferers(investor.referer, toRedistribute, false);* *_updatePoolRewards(toRedistribute, investorAddress, investor.referer);* *accountToInvestorInfo[investorAddress].totalDeposit += toRedistribute;* *totalDepositAmount += toRedistribute;* This inconsistency can lead to mis accounting, since both flows are functionally similar and should operate symmetrically.
Recommendations	Consider updating the deposit flow so that _updatePoolRewards() receives toInvestor (the amount excluding the fee) instead of the full amount. This change will ensure consistent logic across deposit

	and claim flows. However, in the scenario where this is considered as design-choice, it can be acknowledged.
Comments / Resolution	Acknowledged. with comment: User expectation alignment, all tokens should add to the pool reward amount. This issue has been downgraded.

Issue_21	Hardcoded parameters
Severity	**Informational**
Description	The contract contains a number of variables that are completely hardcoded and the contract itself is immutable, locking those values in place for the entire duration of the contract's lifecycle. Most dangerously, the 2 treasuries are such values, introducing potential errors if something happens to those addresses
Recommendations	Consider either upgradability or appropriate owner controlled setter functions when in need of introducing changes to core values
Comments / Resolution	Acknowledged.

Issue_22	curRewards is not reset when pool becomes inactive
Severity	**Informational**
Description	If the curReward of a pool is not reset when the pool becomes inactive, then when it becomes active again, it will immediately start accruing rewards, including for deposits made during its previous active period. This can result in the investor receiving extra rewards, apart from his deposit and subsequent ones. On the other hand, if curReward is reset when the pool deactivates, then upon reactivation, it will start fresh, accruing rewards only from the deposit that reactivates the pool and any deposits made afterwards, as if it were being activated for the first time.
Recommendations	Set the behavior according to what best aligns with the goals of the protocol.
Comments / Resolution	Acknowledged.