BAIL security

On-chain Lottery System

FINAL REPORT

September '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	On-chain Lottery System
Website	
Language	Solidity
Methods	Manual Analysis
Github repository	4debc57ad5ee3aac3cbcd4330d0c1475b2b36241
Resolution 1	commit/de871c4646dfc4b8a5cdabdbcf58df977c77daeb
Resolution 2	commit/574070d636dab8c904b4d1365a1f1ae70ee1b7c7

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High	2	2				
Medium	5	5				
Low	2	1		1		
Informational	5	3		1		1
Governance						
Total	14	10		3		1

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

3. Detection

LottoToken

The LottoToken contract is a simple ERC20 contract with permit functionality. On deployment, it mints an initial supply of tokens to a specified recipient.

No issues found.

PhilippeHelper

The PhilippeHelper contract implements helper functions for packing/unpacking prize and entry data into compact single-slot storage using bit manipulation.

User combinations are stored as bitfields with the leading 96 bits containing the bonus numbers and the rest 160 bits containing the main numbers. Of the 96 bits for bonus numbers, only 64 are used, and the leading 32 bits are actually empty.

Entry IDs are generated from the user combinations by packing the leading empty 32 bits with the current draw ID.

PrizeTiers stores the number of main number matches and the number of bonus number matches. PrizeTierIDs also pack the first 32 bits with the drawID.

Prize tier data is stored in an array with 2 slots dedicated to each main number match, bonus number match combination. This is done by calculating the index as follows.

*index_ = 2 * (mainMatchesCount * (uint256(bonusNumbersCount) + 1) + bonusMatchesCount);*

The first index is used to store the fee in bps for the prize tier, and the second index stores the share in bps of the pot that is assigned to that prize tier.

PhilippeTypes

The PhilippeTypes contract implements bit manipulation operations on the various custom datatypes used in the system, storing bitfields. It implements comparison and counting operations on the bitfield types, which are then used in the main contracts of the system.

Philippe

The Philippe contract is the main contract of the system, handling the sale of tickets, random number drawing, and the distribution of prizes.

Appendix: Participation

Users participate in the lottery by calling the buyEntry function and buying a ticket for a specified combination. Users can participate with multiple tickets at the same time by choosing more numbers than necessary, and the system handles all possible combinations of their selected numbers that can be valid tickets and rewards them later accordingly. If the user chooses N main numbers and each lottery is resolved using R main numbers, the users are essentially sold ${}^{N}C_{R}$ lottery entries at the same time.
The ticket proceeds go towards the pot of the next lottery round. Users are charged stETH for the tickets, but they can also pay with raw ETH if required.

Appendix: Draws

After a minimum time period has elapsed since the start of the lottery, any user can call the requestDraw function with some ETH to invoke the Chainlink VRF system to generate a random number. At this moment, the request ID of the Chainlink request is recorded, and all lottery ticket sales stop. The caller is given a preset amount of reward for advancing the lottery to the next step.
After a few blocks, the Chainlink DON calls the fulfillRandomWords function with a random seed. This seed is then used to generate a series of main and bonus numbers using Floyd's algorithm. The original random seed is hashed over and over again to create new pseudo-random numbers for each iteration of the algorithm.

Appendix: Rewards

The lottery rewards are distributed in a 2-step process. In the first step, ticket holders or delegated entities call the submitEntry function to submit the lottery ticket for rewards. This step calculates the number of matches between the ticket and the result, and decides on prize tiers for the same. For tickets with more main numbers than required, the contract

calculates every possible substring of the user's ticket and signs them up for every possible prize tier they are eligible for.

Once the time window for ticket submissions has passed, users can call the claimPrize function to claim rewards. Each prize tier is allocated a share of the pot, which is then divided amongst all the participants with tickets in that prize tier. Users must claim their prizes within the specified time window; otherwise, they risk losing their rewards and having them roll over to the next lottery.

The jackpot is won when all the main numbers as well as the bonus numbers of a ticket match the draw result. The jackpot is assigned the largest share of the pot, which is then divided between all holders of the jackpot tickets.

Appendix: Attestation

A new round is started only after a jackpot is either confirmed to be present or absent, which is confirmed by the attestation mechanism. If a jackpot entry exists after the draw, anyone can submit that entry using the submitEntry function, which also starts the next round. However, if the jackpot entry is absent, users can call attestNoWinner and put up an attestation collateral in eth. If unchallenged, the validateNoWinner function can be called after a set period of time, repaying the attestation collateral along with a bonus and then starting the next round. If, however, the function attestNoWinner was called fraudulently, anyone can submit the existence of the actual jackpot entry and claim the attestation collateral themselves.

This design allows a fast turnover of the lottery system. The jackpot prize is the largest amount of funds being rolled over, and is thus essential for the next round of the lottery. Thus, it is expected that the moment a winner is discovered, their ticket entry will be submitted, and if not, the attestation mechanism will ensure that the next round will be started at most a certain fixed time period later.

Core Invariants:

INV 1: When buying entries with more main numbers than needed, all possible substrings of the entry are considered during reward calculations.

INV 2: buyEntry stops the moment a randomness request is sent

INV 3: Entries can be submitted only within the timeframe [drawTimestamp, drawTimeStamp +

submitEntryPeriod) and prizes can be claimed only within the timeframe [drawTimeStamp + submitEntryPeriod, drawTimeStamp + winDistributionPeriod). This holds true but separately for both jackpots as well as partial prizes.

INV 4: validateNoWinner can only be called if an attestation was already present and after an attestationPeriod amount of time from the beginning of the attestation.

Privileged Functions

- executeEmergencyShutdown
- initiateShutdown
- withdrawUnallocatedAmount
- withdrawNativeCurrency
- setFeeCollector
- setAttestationCollateral
- setAttestationPeriod
- collectFeesAndYield
- setRequestDrawRewards
- setValidateNoWinnerRewards
- setAttestWinnerRewards
- setReallocateFundsRewards
- setAttestorRewardBps

Issue_01	_submitEth does not repay excess funds
Severity	**Medium**
Description	Users have the option to pay with native ETH when participating in the lotto process. In that case, the user's passed-in ETH amount is wrapped in stETH, and the final balance is checked in the _submitEth function. The issue is that since the stETH-ETH ratio fluctuates, the user will likely send more eth than needed, in order not to have their revert due to an insufficient amount. However, these excess funds are never returned to the user and are instead stored in the contract as yield.
Recommendations	Consider repaying the user the excess stETH.
Comments / Resolution	Fixed by switching from stEth to WBNB.

Issue_02	Negative stETH rebases can lead to temporary halts
Severity	**Medium**
Description	The stETH balance of the contract can go down due to slashing. Since stETH is a rebasing token, it can get rebased down in case of a faulty/malicious validator in the LIDO network. This is a documented risk in their docs. Due to such negative rebases, the current contracts can encounter a revert, since there will be less than expected stETH. In the _transferToAndUpdateAcc function, stETH is transferred out to prize claimants. If insufficient, this function reverts. *if (totalAmount <= Constants.STETH.balanceOf(address(this))) {* *//...* *} else {* *revert IPhilippe.PhilippeInsufficientBalance();* *}* This is also encountered when calling the withdrawUnallocatedAmount function in the admin library, which tries to take out the entire amountToWin, which might not be actually present in case of a negative rebase, reverting that flow. This issue is also encountered whenever rewards are distributed to callers via the _sendRewardsTo function. This function only checks against the pot size, not the actual stETH balance. If the pot size is non-zero, it will try to give out rewards that might not be present in the contract and thus revert. This will block attestation calls as well as rollover calls. The project acknowledges this behaviour for withdrawals/prize claims; however, the same issue also affects attestations, shutdowns, and rollovers, which are all essential parts of the system.
Recommendations	Consider acknowledging if temporary halts are acceptable or if the admin can fund stETH in case of a slashing to keep the system

	operational. Otherwise, consider using wstETH instead, whose balance never gets decreased.
Comments / Resolution	Fixed by switching from stEth to WBNB.

Issue_03	_drawDistinctNumbers with a probabilistic number of iterations is invoked inside fulfillRandomWords, which must complete execution within a pre-agreed amount of gas
Severity	**Low**
Description	Rejection sampling performed inside _drawDistinctNumbers to avoid modulo bias causes the number of iterations to be probabilistic. This means we can't calculate the exact gas required for the execution accurately and will have to take an estimated upper bound. Testing with values can be done to calculate this amount, but it would be a better practice to avoid this form of iteration and place it in a separate function that can be invoked outside of fulfillRandomWords.
Recommendations	Since in certain situations there can be an insufficient amount of gas, consider breaking up the randomness storage and drawing into separate functions, where the draw can be performed by a different user who can send a variable amount of gas. The fulfilrandomness function can try to draw the numbers, but should not fail if it runs out of gas, since that would block the Chainlink call.
Comments / Resolution	Acknowledged.

Issue_04	Multiple updates of drawTimestamp
Severity	**Informational**
Description	The drawTimestamp for the current round is set on requestDraw *function requestDraw() external payable nonReentrant {* *//...* *requestId = _requestRandomnessToChainlinkVRF();* *drawData.requestId = requestId;* *drawData.drawTimestamp = block.timestamp;* *emit DrawRequested(drawId, requestId);* *//...* *}* But then overridden on fulfillRandomWords when the VRF wrapper completes the request. *function fulfillRandomWords(uint256 requestId, uint256[] memory randomNumbers) internal override {* *//...* *drawData.combination = winningCombination;* *drawData.drawTimestamp = block.timestamp;* *emit DrawCompleted(drawId, winningCombination);* *}* This means the initial timestamp set on requestDraw is redundant
Recommendations	The drawTimestamp should either be set on requestDraw or fulfillRandomWords.
Comments / Resolution	Fixed by removing drawTimestamp from requestDraw.

Issue_05	Unnecessary excess currency check for WBNB
Severity	**Informational**
Description	The _depositNative function checks if the submitted amount matches the amount received, within a margin of error. This was necessary earlier, since stEth was used which had a volatile exchange rate to Eth. *require(* *balanceOf >= expectedBalance && balanceOf <= expectedBalance.rawAdd(Constants.MAX_EXCESS_CURRENCY),* *PhilippeFailedNativeDeposit()* *);* Now that the working token has been switched to WBNB, this is unnecessary and can be removed.
Recommendations	With WBNB, the balanceOf will always match expectedBalance, so this check is unnecessary and can be removed.
Comments / Resolution	

RoundSettingsManager

The RoundSettingsManager contract is used to create the configuration of a lottery round, based on certain admin presets and the size of the pot. The contract implements a red-black binary tree to access a tree populated with difficulty values efficiently. One of multiple preset conditions is then chosen based on the difficulty value of the current pot, as calculated by the DifficultyAdjuster contract. The contract also implements the _verifySettingsValidity function, which runs sanity checks on the time restrictions specified in the system, making sure the time windows in the system are positive and sensible.

Core Invariants:

INV 1: Jackpot distribution starts at least 1 minute after jackpot entry period ends.

INV 2: Partial win distribution starts at least 1 minute after the partial win entry period ends.

INV 3: Jackpot distribution deadline is after the partial win period deadline

Issue_06	Partial matching prizes of multicombination jackpot winning entries cannot be claimed in case submitJackpotEntryPeriod is >= partialWinsDistributionPeriod
Severity	**Medium**
Description	Currently, it is not considered that submitJackpotEntryPeriod has to be less than partialWinsDistributionPeriod. This is essential because multicombination jackpot-winning entries can only be submitted for claiming once the submitJackpotEntryPeriod expires. And if by this time the claiming period of partial wins also ends, the winner won't be able to claim it
Recommendations	Ensure that (partialWinsDistributionPeriod - submitJackpotEntryPeriod) is a sufficiently high value
Comments / Resolution	Fixed by requiring partialDistributionPeriod to be at least 1 minute after submitJackpotEntryPeriod.

DifficultyAdjuster

The DifficultyAdjuster contract is used to calculate the appropriate difficulty level of the new round. The difficulty level is estimated primarily from the pot size of the new round and the base pot size of the last round. If the pot amount has increased, the contract searches for a preset difficulty level in the preset red-black binary tree. It then chooses the higher of the nearest difficulty levels if available, and sets it as the difficulty of the current round. If the pot size has decreased, the difficulty is kept the same as that of the last round.

Core Invariants:
INV 1: _retrieveNearestDifficultyLevel retrieves a difficulty value equal to or the nearest higher difficulty level than the one provided. If unsuccessful, the nearest lower value is returned.

Issue_07	Unreachable code - pot amount cannot exceed Uint112.max
Severity	**Informational**
Description	In DifficultyAdjuster::computeDifficultyLevel there's the following if-else block *if (potAmount <= type(uint224).max) {* *// retrieve the nearest existing difficulty level after or before* *// the cast is safe because it was checked just above* *difficulty_ = _retrieveNearestDifficultyLevel(uint224(potAmount));* *} else {* *// retrieve the highest existing difficulty level* *difficulty_ = _retrieveHighestDifficultyLevel();* *}* The issue is that the potAmount is a uint112, if we trace the initial call path Philippe-> getRoundSettings -> computeDifficultyLevel we can see that potAmount was initially a uint112 and implicitly converted to uint256
Recommendations	The if-else block can be removed, since it's redundant.
Comments / Resolution	Fixed following recommendation.

PhilippeAdmin

The PhilippeAdmin library provides owner-only administrative functions for the Philippe lottery system, including emergency shutdown capabilities that allow fund recovery in case of a Chainlink VRF failure/timeout. The contract also implements functions to withdraw unallocated stETH funds in case of a shutdown, as well as recovery of any ETH present in the contract. To trigger a normal shutdown, admins can call the initiateShutdown function, and after the end of the current lottery round, the contract will stop future rounds from being started.

Appendix: Shutdown

When an admin calls the initiateShutdown function, the shutdown state is marked as Initiated in the contract. After this, if a new lottery round creation is attempted, either via submitting a jackpot entry or validating the absence of a jackpot entry via attestations, the system shuts down instead. The round ID is not updated, the shutdown state is set as Validated, and the funds are then made available for recovery.

For the special scenario where intervention is required due to a failure of Chainlink VRF, the executeEmergencyShutdown allows admins to shut down a currently running active round of lottery. The funds are recovered and sent to the specified address, and the shutdown state is marked as validated.

Core Invariants:

INV 1: executeEmergencyShutdown can only be called if the current round has been started at least roundDuration + 24 hours ago.

Issue_08	withdrawUnallocatedAmount can be invoked before the shutdown state is Validated
Severity	**Low**
Description	The admin can withdraw all the funds from the next draw even before the shutdown state becomes Validated. This is flawed as it could cause no amount to be left over to give out caller rewards in the _attestWinnerValidated path
Recommendations	Consider allowing the withdrawUnallocatedAmount to be invoked only after the shutdown state becomes Validated
Comments / Resolution	Fixed following recommendation.

PhilippePrizeDistribution

The PhilippePrizesDistribution library manages the core prize distribution mechanics for the Philippe lottery system, handling entry submissions, prize calculations, and claim processing for both jackpot and partial win scenarios. It implements a two-phase process where players first submit their winning entries after the draw is completed, then claim their proportional share of prizes based on matching numbers and the total number of eligible entries in each prize tier. The library includes automated reallocation functionality that redistributes unclaimed prizes from expired rounds to future draws, and supports both single-combination and multi-combination entries with complex mathematical calculations for prize eligibility and distribution amounts.

Appendix: Reallocation

Every prize tier is assigned a certain share of the whole pot. If there are multiple entries for a tier, that share is split amongst the participants, and if there is no entry for a tier, that share goes unpaid. After the deadline for prize claims, all the unclaimed prize tier funds are reallocated to future rounds.

If the jackpot goes unclaimed, it is immediately reallocated to the next 2 rounds the moment the new round starts. If the other partial win prize tiers go unclaimed, the contract schedules reallocation calls in the future, based on the submission/claim deadlines. Once the deadline passes, anyone can call reallocateUnclaimedAmount directly or internally via other functions to carry out the scheduled reallocations. This moves all unsubmitted and unclaimed prizes to the next immediately available round. This system ensures that all funds in the contracts keep getting reused and are not trapped.

Core Invariants:

INV 1: Submits, claims, and attestations only work after a winning combination has been selected.

INV 2: If a jackpot-winning entry is confirmed present via submission, or confirmed absent via attestation, the next round starts

Issue_09	Jackpot amount per winner can be slashed due to missing validation checks when submitting a jackpot entry
Severity	**High**
Description	Anyone is allowed to submit a jackpot entry on behalf of a player. The player address is retrieved from entryIdToPlayers using the entryId and the player's index, but the player address is not checked/verified (missing address(0) check) *// verify the entry player's address* *prizeDistribution.entryId = entryRepresentation.playerCombination.toEntryId(entryRepresentation.drawId);* *prizeDistribution.**entryPlayerAddress** = s.entryIdToPlayers[prizeDistribution.entryId][entryRepresentation.index];* Afterwards, the player's address is deleted *delete s.entryIdToPlayers[prizeDistribution.entryId][entryRepresentation.index];* However, this only sets the address to address(0). This means anyone can submit the same jackpot entry (i.e., with the entryId & index) again, even though the player address will be address(0) for the subsequent times. The main issue is that submitting a jackpot entry increases the eligibleCombinationsCount for the jackpot prize tier. Since the prize per entry is calculated as $\frac{pot\ prize}{eligibleCombinationsCount}$, this allows a malicious actor to reduce the prize amount per jackpot win by submitting an already submitted entry multiple times, increasing eligibleCombinationsCount

Recommendations	On submitEntry, verify that the player address has not already been deleted (i.e. address(0))
Comments / Resolution	Fixed. Now address(0) is checked in the jackpot winning path as well.

Issue_10	reallocateUnclaimedAmount sends rewards from the current drawId, rustling in possible underflows
Severity	**High**
Description	The _sendRewardsTo function deducts the amountToWin from a round to send stETH to callers to pay them for doing certain actions. Normally, this is always taken from the next round's pot. *DrawId drawId = s.drawId;* *....* *Rewards._sendRewardsTo(msg.sender, drawId, callerRewards);* However, in the reallocateUnclaimedAmount function, this is deducted from the current pot. This changes the amountToWin of an active round. So if this function is called while users are actively claiming prizes, the amountToWin of the round will get reduced, and some users will get fewer prizes than other users even if they hold the same lotto tickets. This makes the prize system unfair. This can also cause underflow inside _calcReallocationAmount since the sum is only guaranteed to be less than amountToWin and not any lower value. An underflow inside _calcReallocationAmount can brick the entire round progression mechanism.
Recommendations	Pay from the next round.
Comments / Resolution	Fixed. Now rewards are paid out from the next round.

Issue_11	Attestor can do a gas griefing attack, delaying round starts
Severity	**Medium**
Description	When validateNoWinner is called, the attestor who had put up some collateral gets repaid via the _tryTransferNativeCurrencyOrSteth function. The issue is that this function first tries to send eth with the entire gasleft() amount, and only then tries wrapping it to stETH. The attestor can implement a receive function in their address and burn all passed in gas. Due to the 63/64 rule, this won't cause an immediate DOS but will allow the attacker to burn up 98% of the passed-in gas. This means for a successful call, the caller must pass in 64x the required amount of gas, such that after burning 98% the remaining gas is still enough to call the stETH wrapping and sending. Thus, the attestor can disincentivize validators, causing a soft DOS until an admin agrees to eat the extra cost to push the transaction through.
Recommendations	Consider using forceSafeTransferAllETH, which uses a strict stipend to prevent this griefing vector.
Comments / Resolution	Fixed following recommendation, call only forwards 100,000 gas.

Issue_12	_attestWinnerValidated can be invoked repeatedly after shutdown, since drawId doesn't get incremented
Severity	**Medium**
Description	In case of jackpot jackpot-winning entry, the submitEntry function always invokes _attestWinnerValidated in case the submitted drawId equals the current drawId. This is based on the assumption that the drawId always gets incremented after an _attestWinnerValidated call. This assumption is incorrect, as if the shutdown state is != NotInitiated, the drawId remains the same. Hence, a user would be able to repeatedly call the _attestWinnerValidated function afterward,s which causes several problems like double counting of assets by incorrect pushes to reallocation queue, jackpot amount double counting in case validateNoWinner was invoked earlier, rewards farming, etc. *if (entryRepresentation.drawId == s.drawId) {* *// if a jackpot winning ticket is submitted for the current round, we attest its presence* *_attestWinnerValidated(entryRepresentation.drawId, roundSettingsManager);*
Recommendations	In case of a shutdown, make sure rollovers are not queued multiple times.
Comments / Resolution	Fixed. _attestWinnerValidated is not called if shutdown is validated.

Issue_13	previousBasePot calculation is not fully accurate, as it doesn't involve the reduced rewards
Severity	**Informational**
Description	The previousBasePot value is calculated as follows *if (DrawId.unwrap(previousDrawId) != 0) {* *// calculate the previous base pot* *previousBasePot = FixedPointMathLib.fullMulDivUp(* *previousDrawData.amountToWin, Constants.BPS_DIVISOR, s.roundSettings.potAllocationBps* *);* But this is not entirely accurate, as caller and attestor rewards were further reduced from the allocation fraction to obtain amountToWin. The current basePot amount being compared did not have those numbers reduced yet.
Recommendations	Consider acknowledging this issue, since the difference will be small under normal operating conditions.
Comments / Resolution	Acknowledged.

PhilippeRewards

The PhilippeRewards library manages gas cost reimbursement and incentive mechanisms for users who perform critical protocol operations such as requesting draws, validating no-winner scenarios, attesting winners, and reallocating unclaimed funds. It calculates rewards based on current network gas fees (capped at configurable maximum base fees) multiplied by adjustable multiplier rates, with rewards paid from the next round's prize pool in stETH tokens. The library includes administrative functions to configure reward settings for different operations and implements validation checks to ensure reasonable gas fee caps and multiplier ranges to prevent misconfigurations while maintaining protocol sustainability

Core Invariants:

INV 1: Rewards sent via _sendRewardsTo are taken from the next available pot.

INV 2: gas fee reimbursement in _calcRewards is capped to a specified maxBaseFee limit.

No issues found

PhilippeUtils

The PhilippeUtils library provides essential utility functions for the Philippe lottery system, managing stETH token transfers with balance tracking, mathematical calculations for prize distributions, and automated fee collection. It handles the protocol's storage access, safe token operations, accounting for stETH rebasing, and transfers accumulated fees to the designated collector with optional callback functionality.

Core Invariants:

INV 1: Only stEth tokens above the recorded balanceAccumulator amount are regarded as yield and processed accordingly.

INV 2: balanceAccumulator always gets updated when stEth is transferred in or out of the system by the movement amount.

PhilippeConstants

The PhilippeConstants library defines system-wide constants for the Philippe lottery protocol, including the stETH token address, basis points divisor, bit field layouts for combinations and prize tiers, attestation limits, gas cost estimates, and EIP712 typehashes for delegation signatures.

Issue_14	Dummy values present for gas costs
Severity	**Informational**
Description	Variables REQUEST_DRAW_GAS_COST, VALIDATE_NO_WINNER_GAS_COST, ATTEST_WINNER_GAS_COST, and REALLOCATED_PRIZE_GAS_COST contain placeholder values that don't reflect the actual gas costs. These should be updated to match actual on-chain gas costs before deployment.
Recommendations	Update the variable to match actual gas costs when calling those functions.
Comments / Resolution	Fixed. Dummy values have been updated. But it would be ideal if the values are modifiable (at the expense of a slight gas cost increase) in order to incorporate any change in gas pricing.