PokPok Protocol

Smart Contract Audit Report

20th August 2025

FAILSAFE

Table of Contents

Executive Summary 2

Project Details 3
- Structure & Organization of Audit Report . . . 3

Project Goals 4

Audit Methodology 5
- In-scope Files . . . 5

Summary of Findings 6
- Finding 1: Missing Validation for Stuck Harvest in resetHarvestState . . . 7
- Finding 2: Desynchronization and Silent Failure Risks in Batch Harvest and State Update Logic . . . 9
- Finding 3: Silent Failures in NFT Approval Operations . . . 11
- Finding 4: Parameter and Initialization Validation Deficiencies in Constructor and Admin Updates . . . 12
- Finding 5: Missing Events for Important State Changes . . . 14
- Finding 6: Front-Running and Timing Attacks on Share Price Updates . . . 16

Disclaimer 17

Executive Summary

The objective of this project is to perform a comprehensive security and functionality audit of the PokPok Protocol smart contracts. PokPok Protocol is a DeFi protocol that enables users to engage in options trading and yield farming through innovative vault mechanisms and chicken game dynamics. The protocol features sophisticated position management, harvest operations, and risk management systems designed to optimize returns while maintaining security. Given the high value and irreversible nature of blockchain transactions, any vulnerability in these contracts could result in significant financial loss and reputational damage. This audit aims to identify critical, major, and minor issues, as well as provide optimization recommendations to enhance code efficiency, maintainability, and transparency.

Throughout the audit process, the PokPok Protocol team demonstrated exceptional commitment to security, responding to findings with remarkable speed and implementing fixes within hours of receiving critical vulnerability reports. Their proactive security posture and rapid response capabilities reflect a mature security culture and strong operational discipline. The team's ability to quickly validate, acknowledge, and remediate security issues showcases their technical expertise and dedication to maintaining the highest security standards.

Project Details

Project	PokPok Protocol
URL	https://pokpok.io/
Source Code	https://github.com/PokPokProtocolOrg/Protocol_contract
Initial Commit	0d5cc57812027037f8b77824e2261d732ada9d99
Final Commit	c957b8a950ad67e3883c06791e4d71925d0617f6
Timeline	Initial Report - 22nd July 2025 - 30th July 2025
	Final Report - 30th July 2025 - 20th August 2025

Structure & Organization of Audit Report

Issues are tagged as “Open”, “Acknowledged”, “Partially Resolved”, “Resolved” or “Closed” depending on whether they have been fixed or addressed.

- Open: The issue has been reported and is awaiting remediation from developer team.
- Acknowledged: The developer team has reviewed and accepted the issue but has decided not to fix it.
- Partially Resolved: Mitigations have been applied, yet some risks or gaps still remain.
- Resolved: The issue has been fully addressed and no further work is necessary.
- Closed: The issue is deemed no longer pertinent or actionable.

Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical	The issue affects the Smart Contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.
High	The issue affects the ability of the Smart Contract to compile or operate in a significant way.
Medium	The issue affects the ability of the Smart Contract to operate in a way that doesn’t significantly hinder its behavior.
Low	The issue has minimal impact on the Smart Contract’s ability to operate.
Info	The issue is informational in nature and does not pose any direct risk to the Smart Contract’s operation.

Project Goals

1. Security Assurance

Ensure the PokPok Protocol smart contracts are free of critical vulnerabilities and follow industry-standard security best practices (e.g., SWC Registry, OpenZeppelin guidelines).

2. Functional Correctness

Verify that the contracts behave as intended according to the PokPok Protocol specification, including proper handling of edge cases and failure modes for position management, harvesting operations, and vault interactions.

3. Gas Optimization

Review the contracts for inefficient logic or high gas-consuming operations and provide suggestions to optimize transaction costs for users and protocol operators.

4. Access Control & Privileges

Analyze the roles and permissions within the contracts to prevent unauthorized actions, privilege escalations, or accidental lockouts, with a focus on the allowlist, admin roles, and pausing mechanisms.

5. Upgradability & Maintainability

Evaluate the contracts' upgradeability and maintainability, especially regarding proxy patterns, modular architecture, and future extensibility.

6. Compliance & Documentation

Ensure that the contracts are well-documented, follow Solidity development conventions, and provide clear, accessible documentation for future developers and auditors.

7. Reporting & Remediation Guidance

Deliver a detailed audit report categorizing all findings and recommending steps for remediation, along with a final verification round post-fix to ensure all issues are addressed.

Audit Methodology

FailSafe employs a multi-layered approach to Smart Contract security audits:

Threat Modelling: We identify critical assets, enumerate potential threats, assess vulnerabilities, and prioritize risks based on severity and impact.

Manual Code Review: Our experts conduct a detailed, line-by-line review of the code, analyzing business logic, access controls, gas efficiency, and external dependencies.

Functional Testing: Using frameworks like Hardhat and Foundry, we perform comprehensive functional and integration tests to ensure correct and secure Smart Contract behavior.

Fuzzing & Invariant Testing: Advanced techniques such as fuzzing and invariant testing are used to uncover hidden vulnerabilities and verify Smart Contract consistency under diverse scenarios.

Edge Case Analysis: We rigorously test for extreme inputs, exception handling, concurrency, and non-standard scenarios to ensure robust Smart Contract performance.

Reporting & Recommendations: Our reports clearly describe each issue, its impact, location, root cause, and provide actionable remediation steps and best practice guidelines.

Remediation Support: We work closely with your team to implement and validate fixes, followed by a final assessment to confirm all issues are resolved.

FailSafe's process ensures your Smart Contracts are secure from initial deployment through ongoing operation, providing proactive and comprehensive protection.

In-scope Files

- contracts/PokPokVault.sol
- contracts/PokPokVaultFactory.sol

Summary of Findings

Severity	Total	Open	Acknowledged	Partially Resolved	Resolved	Closed
Critical	-	-	-	-	-	-
High	1	-	-	-	1	-
Medium	2	-	-	-	2	-
Low	2	-	-	1	1	-
Info	1	-	1	-	-	-
Total	6	0	1	1	4	0

#	Findings	Severity	Status
1	Missing Validation for Stuck Harvest in resetHarvestState	High	Resolved
2	Desynchronization and Silent Failure Risks in Batch Harvest and State Update Logic	Medium	Resolved
3	Silent Failures in NFT Approval Operations	Medium	Resolved
4	Parameter and Initialization Validation Deficiencies in Constructor and Admin Updates	Low	Partially Resolved
5	Missing Events for Important State Changes	Low	Resolved
6	Front-Running and Timing Attacks on Share Price Updates	Info	Acknowledged

Finding 1: Missing Validation for Stuck Harvest in resetHarvestState

Severity: High

Status: Resolved

Source: Contract: PokPokVault.sol Function: resetHarvestState (lines 729-735)

Description:

The `resetHarvestState` function allows resetting an in-progress harvest without validating if it's truly 'stuck' (e.g., no timeout check on `_harvestInProgressTimestamp` or confirmation of pending `_pendingMaturedIds`). This enables arbitrary calls by VAULT_MINTER_ROLE, potentially dropping matured positions prematurely, causing lost yields or state desync, deviating from info.doc's 'efficient batch processing'.

Impact:

Allows premature dropping of matured positions, leading to lost yields, inaccurate TVL, and state desynchronization; practical for minter misuse, potentially locking assets until manual intervention.

Code:

```
function resetHarvestState() external onlyVaultMinter {
  if (_harvestInProgressTimestamp == 0) revert Vault_HarvestNotInProgress();
  delete _pendingMaturedIds;
  _nextBatchIndex = 0;
  _harvestInProgressTimestamp = 0;
}  // No stuck check
```

Proof of Concept:

1. Call harvestMaturedOptions to initiate a harvest with more pending matured IDs than MAX_BATCH_SIZE (e.g., >15), processing only the first batch and leaving the rest in progress.
2. In a subsequent transaction, before calling harvestMaturedOptions again to continue batches, call resetHarvestState as minter.
3. Observe `_pendingMaturedIds` cleared and harvest timestamp reset, dropping remaining pending harvests without verifying stuck status.

Remediation:

Add validation like require(block.timestamp - _harvestInProgressTimestamp > 1 hours && _pendingMaturedIds.length > 0, "Not stuck"); additionally, loop through `_pendingMaturedIds` to

check for failed/stuck IDs (e.g., try querying maturity via `chickenData.getChicken` and revert if any can be harvested); emit `ResetExecuted(block.timestamp, _pendingMaturedIds.length)` ;

Developer Response:

Fix commit hash: c957b8a950ad67e3883c06791e4d71925d0617f6.

Auditor Response:

Verified the fix.

Finding 2: Desynchronization and Silent Failure Risks in Batch Harvest and State Update Logic

Severity: Medium

Status: Resolved

Source: Contract: PokPokVault.sol Functions: manualHarvest, continueHarvest (via harvestMaturedOptions)

Description:

Both the `manualHarvest` and `continueHarvest` functions use a try-catch pattern for external contract calls (such as `proxy.harvestChicken` and `chickenData.getChicken`) but allow the contract to continue operating even if key internal state updates are skipped due to failures in these calls. This creates a risk of desynchronization between the vault's on-chain accounting and the actual state of positions, as well as "silent failures" with minimal or no event logging, especially in batch operations. Desynchronization occurs when `proxy.harvestChicken` succeeds but `chickenData.getChicken` fails, leaving inflated `investedAssets`, outdated `activeChickenIds`, and inaccurate VaR. Silent failures happen in `continueHarvest` with empty `else {}` blocks emitting no events.

Impact:

Inflated metrics (e.g., VaR, TVL) can block new trades or allow unfair withdrawals; prolonged vault inactivity requires manual fixes; users may see incorrect share pricing, eroding trust.

Code:

```
try proxy.harvestChicken(tokenId) {
  try chickenData.getChicken(tokenId) returns (IChickenData.Chicken memory
      chicken) {
    if (chicken.settled && block.timestamp >= chicken.maturityTimestamp) {
      // normal flow
    } else {
      revert Vault_InvalidStateAfterHarvest(tokenId);
    }
  } catch {
    revert Vault_DataFetchFailedAfterHarvest(tokenId);
  }
} catch {}
```

Proof of Concept:

- Single Position: One `manualHarvest` succeeds externally, fails internally—vault holds 'ghost' position, VaR inflated.
- Batch Failure: `continueHarvest` skips positions due to `getChicken` failures—metrics stale, no events signal failures.

Remediation:

After a successful external harvest, if internal data fetch fails, revert the transaction to avoid desync. Emit events on approval failures, or revert if approval is critical. Periodically audit approvals and provide on-chain methods to check/repair approval state.

Developer Response:

Fix commit hash: c957b8a950ad67e3883c06791e4d71925d0617f6.

Auditor Response:

Verified the fix.

Finding 3: Silent Failures in NFT Approval Operations

Severity: Medium

Status: Resolved

Source: Contract: PokPokVault.sol Functions: constructor, updateAddresses

Description:

Try-catch blocks around `chickenNft.setApprovalForAll` in the constructor and `updateAddresses` function swallow errors with no revert or event emission. Failures in setting or revoking NFT approvals for protocol contracts can leave the vault in an inconsistent or insecure state (e.g., unable to transfer NFTs, or old protocols retaining approval).

Impact:

Failed approvals may block future harvesting or order placement; old protocols may retain approval for vault-owned NFTs, risking unauthorized transfers; could render automated harvesting unreliable, increasing operational overhead.

Code:

```
// constructor
try chickenNft.setApprovalForAll(_protocolAddr, true) {} catch {}

// updateAddresses
if (oldProtocol != address(0) && oldProtocol != _newProtocol && address(
    chickenNft) != address(0)) {
  try chickenNft.setApprovalForAll(oldProtocol, false) {} catch {}
}
if (oldNft != address(0) && oldNft != _chickenNft && _protocol != address(0)) {
  try IERC721(oldNft).setApprovalForAll(_protocol, false) {} catch {}
}
if (address(chickenNft) != address(0) && _protocol != address(0)) {
  try chickenNft.setApprovalForAll(_protocol, true) {} catch {}
}
```

Remediation:

Emit events on approval failures, or revert if approval is critical for subsequent operations. Periodically audit approvals and provide on-chain methods to check/repair approval state.

Developer Response:

Fix commit hash: c957b8a950ad67e3883c06791e4d71925d0617f6.

Auditor Response:

Verified the fix.

Finding 4: Parameter and Initialization Validation Deficiencies in Constructor and Admin Updates

Severity: Low

Status: Partially Resolved

Source: Contract: PokPokVault.sol Functions: constructor, createVault (in PokPokVaultFactory.sol), updateMinDeposit (line 680), updateActivationThreshold (line 685), updateRiskParameters (line 701), updateFeeParameters (line 708), updateMaxCapacity (line 723), and related admin functions like rescueETH (line 818)

Description:

Constructor, factory, and admin update functions lack tight bounds/edge checks for risk params and thresholds (e.g., `_maxVaR` up to 100% vs doc's 20-50%, no lowers like thresholds >0; update functions allow zero or extreme values like `minDeposit=0` enabling dust attacks or `maxCapacity` below current TVL trapping funds), with mismatched errors and no factory mirroring—allowing unsafe deployments and post-deployment changes that overexpose TVL or create limbo states.

Impact:

Overexposure risks total TVL loss; permanent for deployed vaults, contradicting doc's risk limits; enables spam via dust deposits, disables safeguards, or locks funds—practical for admin compromise, high exploitability in role-based attacks.

Code:

```
// Constructor validation (loose)
if (_maxVaR > MAX_BASIS_POINTS || ... ) revert Vault_InvalidFeePercentage();
    // Treats risk as fees, allows high values

// updateMinDeposit - no zero check
function updateMinDeposit(uint256 _minDeposit) external onlyPokAdmin {
  minDeposit = _minDeposit;  // Allows 0 or max uint
}

// updateRiskParameters - no lower bound
if (_maxVaR > MAX_BASIS_POINTS || _stdSizePct > MAX_BASIS_POINTS) revert;
maxVaR = _maxVaR;  // Allows 0, disabling protection
```

Remediation:

Add `require(_maxVaR <=5000 && _maxVaR >=1000, 'Invalid VaR')`; use `Vault_InvalidRiskParam`; enforce `_activationThreshold > minValue`; mirror in factory `createVault`; for updates, add checks like `require(_minDeposit > 0 && _minDeposit < maxCapacity, "Invalid min deposit")`; implement similar for all.

Auditor Response:

The team has successfully implemented comprehensive parameter validation across all admin functions and the factory, with proper bounds checking and event emission. The validation infrastructure is robust and eliminates the core security risks we identified.

Our only remaining observation is that the actual parameter bounds are more permissive than initially recommended, maxVaR caps at 80% rather than 50%, and performance fees at 100% rather than 50%. These are policy decisions around risk tolerance rather than security vulnerabilities, and the current bounds still provide reasonable protection against misconfiguration.

Finding 5: Missing Events for Important State Changes

Severity: Low

Status: Resolved

Source: Contract: PokPokVault.sol Function: updateMinDeposit, updateRiskParameters, updateFeeParameters, updateMaxCapacity, updateAddresses, resetHarvestState, setApprovalForAll, emergencyWithdrawToken, emergencyWithdrawNFT, rescueETH (note: some like updateActivationThreshold and pause/unpause have partial/conditional emits)

Description:

Many admin and emergency functions modify critical state (e.g., parameters, resets, approvals, token rescues) without emitting events, reducing off-chain transparency and auditability. While some (e.g., `updateActivationThreshold` emits on change, pause emits via OZ) have coverage, others like param updates and resets are silent, deviating from best practices for tracking changes in role-based systems.

Impact:

Undetected changes (e.g., fee hikes or resets) could enable abuse if admin compromised; hinders monitoring, but no direct exploit without other vulns—partial emits mitigate some.

Code:

```
// Example without emit: updateFeeParameters
function updateFeeParameters(
  uint256 _performanceFeePct,
  // ...
) external onlyPokAdmin {
  // Assignments, no emit
}

// Example with conditional emit: updateActivationThreshold
function updateActivationThreshold(uint256 _activationThreshold) external
    onlyPokAdmin {
  activationThreshold = _activationThreshold;
  // ...
  if (shouldBeActive) { emit VaultActivated(tvl); } else { emit
      VaultDeactivated(tvl); }
}
```

Remediation:

Add emits like `emit FeesUpdated(_performanceFeePct, etc.)`; for reset, `emit HarvestReset(block.timestamp)`; ensure consistent logging across all state mods, building on existing conditional ones.

Developer Response:

Fix commit hash: c957b8a950ad67e3883c06791e4d71925d0617f6.

Auditor Response:

Verified the fix.

Finding 6: Front-Running and Timing Attacks on Share Price Updates

Severity: Info

Status: Acknowledged

Source: Contract: PokPokVault.sol Function: _updateSharePrice, deposit, withdraw, harvestMaturedOptions

Description:

Timing attacks allow front-running price updates (e.g., deposit before harvest profit pump, withdraw after) to capture dilution gains, with unchecked floors and interactions amplifying in high-MEV environments.

Impact:

User dilution (0.1-1% per tx), bot profits scaling with vault activity—higher on L2s.

Code:

```
// Price update (pumpable)
newSharePrice = (totalVaultValue * PRECISION) / currentTotalSupply;

// Harvest adds profit
_liquidAssets += valuation;  // Pumps TVL before post-update
```

Developer Response:

We acknowledge the issue and plan to address it in a future update.

Disclaimer

This audit report (“Report”) is provided by FailSafe (“Auditor”) for the exclusive use of the client (“Client”). The audit scope is limited to a technical review of the Smart Contract code supplied by the Client. This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without FailSafe’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts FailSafe to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. FailSafe’s position is that each company and individual are responsible for their own due diligence and continuous security. FailSafe’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by FailSafe is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, FAILSAFE HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, FAILSAFE SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, FAILSAFE MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

WITHOUT LIMITATION TO THE FOREGOING, FAILSAFE PROVIDES NO WARRANTY OR DISCLAIMER UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER FAILSAFE NOR ANY OF FAILSAFE'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. FAILSAFE WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT FAILSAFE'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST FAILSAFE WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF FAILSAFE CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST FAILSAFE WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.