



SECURITY AUDIT

Podha

Protocol Backend

CLIENT

Podha Protocol

REVIEW

September 2025

DOCUMENT

Document information

Project	Podha, Protocol Backend
Client	Podha Protocol
Platform	Not recorded in the original report
Language	Not recorded in the original report
Repository	parth26nath/Podha_Backend
Commit	Not recorded in the original report
Methods	Manual code review, static analysis, best practices check
Review	1 to 11 September 2025
Report date	21 September 2025

Document history

VERSION	DATE	CHANGE
1.0	21 September 2025	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	L-01 Management Fee Logic Could Be Automated	8
	L-02 Early Exit Fees Logic to Match Documentation	9
	L-03 Multiple Yield Tiers Can Provide More Flexibility	10
	L-04 Admin Functions and Centralization	11
07	Conclusion	12
A	Appendix: vulnerability checklist	13
	Disclaimer	14

Executive summary

About Podha

Podha Protocol is a multi-strategy asset management and yield generation platform. It enables users to allocate capital in a permissionless manner to a diversified yield engine, earning sustainable returns and Drop Points over time. Its strategies include intra-exchange arbitrage, funding rate capture, delta-neutral positioning and selective directional trades.

Summary of the review

Safe Edges reviewed the Podha Protocol backend between 1 and 11 September 2025. Four Low severity issues were identified, covering management fees that are applied manually, early exit fees that do not match the documentation, a single deposit tier, and admin functions held by a single admin. Podha acknowledged all four and plans to address them in future updates.

Findings by severity



Findings by status

STATUS	ISSUES
● Acknowledged	4

Scope

Repository	parth26nath/Podha_Backend
Commit	Not recorded in the original report
Platform	Not recorded in the original report
Language	Not recorded in the original report

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Manual code review, static analysis, best practices check.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
L-01	Management Fee Logic Could Be Automated	Low	● Acknowledged
L-02	Early Exit Fees Logic to Match Documentation	Low	● Acknowledged
L-03	Multiple Yield Tiers Can Provide More Flexibility	Low	● Acknowledged
L-04	Admin Functions and Centralization	Low	● Acknowledged

Detailed findings

L-01 Management Fee Logic Could Be Automated

Severity **Low** Status ● Acknowledged

OBSERVATION

Currently, management fees for users withdrawing before the 30-day lock period are applied manually. Automating this process could improve consistency and reduce manual effort.

RECOMMENDATIONS

- Automate fee deduction within smart contract or backend service.
- Ensure fees are calculated and applied consistently.
- Consider flexible parameters for adjusting fees in the future.

CLIENT RESPONSE

Automation is planned in future updates.

EXAMPLE IMPLEMENTATION (JAVASCRIPT)

```
const MANAGEMENT_FEE_RATE = 0.01; // 1%
function calculateManagementFee(withdrawAmount) {
  return withdrawAmount * MANAGEMENT_FEE_RATE;
}
function processWithdrawal(user, withdrawAmount) {
  const fee = calculateManagementFee(withdrawAmount);
  const netAmount = withdrawAmount - fee;
  console.log(`Processed withdrawal for ${user.name}: Gross=${withdrawAmount}, F
```

L-02 Early Exit Fees Logic to Match Documentation

Severity **Low** Status ● Acknowledged

OBSERVATION

Users can withdraw principal at any time, but documentation specifies a 2.5% early exit fee. Implementing this logic aligns platform with documentation.

RECOMMENDATIONS

- Add smart contract or backend logic to apply early exit fees.
- Display early exit options and corresponding fees clearly on frontend.

CLIENT RESPONSE

Early exit fee logic can be added in upcoming updates.

EXAMPLE IMPLEMENTATION (JAVASCRIPT)

```
const EARLY_EXIT_FEE_RATE = 0.025; // 2.5%
function isEarlyWithdrawal(depositDate, lockPeriodDays) {
  const now = new Date();
  const lockEndDate = new Date(depositDate);
  lockEndDate.setDate(lockEndDate.getDate() + lockPeriodDays);
  return now < lockEndDate;
}
function processEarlyExit(user, withdrawAmount, depositDate, lockPeriodDays) {
  let fee = 0;
  if (isEarlyWithdrawal(depositDate, lockPeriodDays)) {
    fee = withdrawAmount * EARLY_EXIT_FEE_RATE;
  }
  const netAmount = withdrawAmount - fee;
  console.log(`Early exit withdrawal for ${user.name}: Fee=${fee}, Net=${netAmount}`);
}
```

L-03 Multiple Yield Tiers Can Provide More Flexibility

Severity **Low** Status ● Acknowledged

OBSERVATION

Only a single deposit tier (\$50 minimum) is supported. Multiple tiers could offer more flexibility.

RECOMMENDATIONS

- Implement multiple yield tiers (50, 150, 500, 1500).
- Ensure smart contract logic supports tiered rewards.
- Show tiers clearly on frontend.

CLIENT RESPONSE

Multiple categories will be introduced in future updates.

EXAMPLE IMPLEMENTATION (JAVASCRIPT)

```
const yieldTiers = [
  { minDeposit: 50, yieldRate: 0.03 },
  { minDeposit: 150, yieldRate: 0.04 },
  { minDeposit: 500, yieldRate: 0.05 },
  { minDeposit: 1500, yieldRate: 0.06 },
];

function getYieldRate(depositAmount) {
  let applicableTier = yieldTiers.slice().reverse().find(tier => depositAmount >
  return applicableTier ? applicableTier.yieldRate : 0;
}

function calculateYield(depositAmount) {
  const rate = getYieldRate(depositAmount);
  return depositAmount * rate;
}
```

L-04 Admin Functions and Centralization

Severity **Low** Status ● Acknowledged

OBSERVATION

Some critical functions are handled by a single admin, creating centralization risk.

RECOMMENDATIONS

- Automate routine admin tasks.
- Implement multi-sig approvals for sensitive actions.
- Assign multiple roles for actions (Fee Manager, Withdrawal Approver, Auditor).
- Maintain separate audit trails.

CLIENT RESPONSE

Automation and decentralization are planned for future updates.

EXAMPLE IMPLEMENTATION (JAVASCRIPT)

```
const roles = {
  feeManager: [],
  withdrawalApprover: [],
  auditor: []
};

const requiredApprovals = 2;
let approvals = {};

function assignRole(role, adminId) {
  if (!roles[role].includes(adminId)) {
    roles[role].push(adminId);
  }
}

function approveAction(role, adminId, actionId) {
  if (!approvals[actionId]) approvals[actionId] = {};
  if (!approvals[actionId][role]) approvals[actionId][role] = new Set();
  approvals[actionId][role].add(adminId);
  if (approvals[actionId][role].size >= requiredApprovals) {
    executeAction(actionId, role);
  }
}

function executeAction(actionId, role) {
  approvals[actionId][role] = new Set(); // reset
}
```

Conclusion

Podha Protocol is functional with a solid foundation. Implementing the suggested improvements will:

- Automate management and withdrawal processes
- Introduce early exit fees to match documentation
- Expand yield tiers for flexibility
- Streamline admin tasks with multi-sig, multi-role, and audit trails

APPENDIX A

Vulnerability checklist

Vulnerability classes checked during this review, as listed in the original report.

✓ Access Management	✓ Compiler version not fixed
✓ Arbitrary write to storage	✓ Address hardcoded
✓ Centralization of control	✓ Divide before multiply
✓ Ether theft	✓ Integer overflow/underflow
✓ Improper or missing events	✓ ERC compliance
✓ Logical issues and flaws	✓ Dangerous strict equalities
✓ Arithmetic Computations Correctness	✓ Tautology or contradiction
✓ Race conditions/front running	✓ Return values of low-level calls
✓ SWC Registry	✓ Missing Zero Address Validation
✓ Re-entrancy	✓ Private modifier
✓ Timestamp Dependence	✓ Revert/require functions
✓ Gas Limit and Loops	✓ Multiple Sends
✓ Exception Disorder	✓ Using suicide
✓ Gasless Send	✓ Using delegatecall
✓ Use of tx.origin	✓ Upgradeable safety
✓ Malicious libraries	✓ Using throw

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS