RISK AUDIT

for

PB&J CONSULTING

on

Jan 26, 2026

FIDESIUM

Executive Summary

Report

12
/100

TOTAL
Low risk
Jan 26, 2025

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on Lava **contracts**

Contract Link: https://sepolia.basescan.org/address/

`0xb45ae157bedf5eae20fa0d24cb12f4e91aabdf55#code`

Revised contract Link: https://sepolia.basescan.org/address/

`0x135d7797452405358ea9d38d156603F73275B99b`

Issue Summary

Critical	High	Medium	Low	Info
~~1~~ 0 Issues	0 Issues	2 Issues	2 Issues	1 Issues

Caveats

PBJ's codebase is generally well written, but does incur a handful of flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vulnerability	Description	Risk	Probability	Status
Irrecoverable State	Liquidity Pool depletion can cause an irrecoverable state, permanently bricking the protocol	Critical	Low	Acknowledged
Centralization	The `owner` has significant modification rights over the contracts and their state.	Medium	Medium	Acknowledged
Possible blocking due to undiscovered errors	The `rescueETH` function can be blocked if undiscovered accounting bugs cause `totalPendingPrizes` to deviate.	Medium	Low	Active
Reliance on call	The contract relies on `call`.	Low	Low	Acknowledged
Precision Loss	The cotnract relies on division.	Low	Low	Acknowledged
Gas Inefficiency: Compound Interest calculated in loop	The contract uses a loop to compute compound interest	Info	Info	Active

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Smart Contract Risks

Risk summary: 17

The contracts are well written, and secure with only a few minor issues..

Vulnerabilities Critical

Irrecoverable State

Vulnerability severity: **Critical**

Vulnerability probability: **Low**

Liquidity Pool depletion can cause an irrecoverable state, permanently bricking the protocol

LP tokens deplete geometrically: Round 1 → 50%, Round 2 → 25%, Round 3 → 12.5%

`checkAndExecuteRug` removes liquidity

```
(uint256 lavaRemoved, uint256 ethRemoved) = router.removeLiquidityETH(
    address(this),
    lpToRemove,
    0,
    0,
    address(this),
    block.timestamp + 300
);
```

However, `refillLiquidity` relies on `IUniswapV2Pair(pair).sync();`. Sync is defined within uniswap as:

```
function sync() external lock {
    _update(
        IERC20(token0).balanceOf(address(this)),
        IERC20(token1).balanceOf(address(this)),
        reserve0,
        reserve1
    );
}
```

This correctly refills the reserves but **does not** mint new lptokens. Eventually, lpBalance will reach 0, and with no way to refresh the pool, the contract will permanently brick.

`checkAndExecuteRug()` requires `lpBalance > 0` - when LP ≈ 0, function reverts

Recommendations:

1. **Primary fix:** Modify `refillLiquidity()` to call `router.addLiquidityETH()` instead of `sync()`
2. **Emergency recovery:** Modify `addLiquidity()` to remove the `require(!tradingEnabled)` check, OR add a new `emergencyAddLiquidity()` function that can only be called when LP balance is critically low
3. **Critical:** Implement ETH retention mechanism - currently contract has no ETH to add liquidity (all ETH goes to winners)

Action Taken:

Remediated, The event is 60-70 rounds in the future, however a `manualAddLiquidity` was added to allow for resolution.

Vulnerabilities High

Current scan highs `Clear`

During this scan no high security vulnerabilities were identified. The assessment covered all key components of the project, including smart contract logic, access controls, and potential attack vectors. While no critical issues were found, we recommend ongoing security monitoring and best practices to maintain the integrity and resilience of the system.

Vulnerabilities Medium

Centralization

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `owner` has significant modification rights over the contracts and their state.

Recommendations:

Ensure owner is a well managed multisig

Actions Taken:

Owner will be a well managed multisig

Possible blocking due to undiscovered errors

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The `rescueETH` function can be blocked if undiscovered accounting bugs cause `totalPendingPrizes` to deviate.

Recommendations:

Implement an emergency, timelocked, owner only rescue function which bypasses accounting checks

Vulnerabilities Low

Precision Loss

Vulnerability severity: **Low**

Vulnerability probability: **Low**

The conract relies on division.

```
currentFloor = (currentFloor * (10000 + rate)) / 10000;
```

Over thousands of iterations this will slowly drift from actual values due to truncation

Recommendations:

Use a mathematical compound formula instead of iterative multiplication, or track fractional parts separately.

Actions Taken:

This precision loss will be negligible under real conditions

Reliance on call

Vulnerability severity: **Low**

Vulnerability probability: **Low**

The contract relies on `call`.

Low level calls will copy any amounts of bytes to local memory, allowing for gas griefing via a returnbomb

Recommendations:

Use nomad-xyz/excessivelysafecall instead of call

Actions Taken:

There are no concrete attack vecotrs and little discovered incentive for a caller to, in effect, grief themselves. That said the exploit exists in theory

Vulnerabilities Info

Gas Inefficiency: Compound Interest calculated in loop

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The contract uses a loop to compute compound interest

```
for (uint256 i = 0; i < periodsToProcess; i++) {
    uint256 period = lastUpdatePeriod + 1 + i;
    uint256 rate = _getRateForPeriod(period);
    currentFloor = (currentFloor * (10000 + rate)) / 10000;
}
```

The gas cost will be linear to the number of iterations, at cap of 500, this would result in ~10000 gas per iteration

Recommendations:

Apply the compound interest formula for a static gas cost reduction of ~98% at cap of 500. Since 10500^500 will overflow uint 256, this will require careful SafeMath-style computation with chunking (e.g. 50 periods at a time). It is also worth noting that while correct, the result will differ from sequential rounding, and will require testing.

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.