LAF 审计报告

Smart Contract Security Audit Report

Prepared by: Audit911**Date:** 2025年12月03日

Disclaimer & Confidentiality

> 本安全审计报告文件可能包含机密信息。
> 本文档包含潜在的漏洞和恶意代码分析。由于 Owner 权限已弃权，发现的漏洞可能已无法修复，建议投资者仅作风险评估参考。
>
> **Audit911** 已按照行业最佳实践对该智能合约进行了分析。然而，本审计并不保证被审计智能合约的绝对安全性。建议所有投资者/用户自行进行尽职调查。

1. Project Overview (项目概览)

1.1 Audit Scope (审计范围)

项目	描述
项目名称	Computility (LAF)
平台	EVM Compatible (Ethereum/BSC 等)
语言	Solidity (^0.8.20)
智能合约代码	LAF.sol, Staking.sol
审计日期	2025年12月03日

1.2 Introduction (简介)

本次审计旨在评估 `LAF` 代币合约与 `Staking` 质押合约的安全性、业务逻辑稳健性以及经济模型的潜在风险。由于项目方已放弃所有权（Owner Renounced），审计重点在于代码中不可变的逻辑漏洞及资产安全风险。

1.3 Project Background (项目背景)

该项目包含一个名为 LAF 的 ERC20 代币和一个相关的质押（Staking）合约。用户需通过 `Staking` 合约存入 USDT 参与，合约会自动购买 LAF 并添加流动性。项目声称通过质押产生收益，但实际上包含复杂的资产回收机制。

2. Audit Summary (审计总结)

根据标准审计评估，客户的智能合约安全性评级为 **[Insecure / 极不安全]**。

2.1 Vulnerability Count (漏洞统计)

严重程度	数量
🔴 **Critical (严重)**	1
🟠 **High (高危)**	1
🟡 **Medium (中危)**	1
🟢 **Low (低危)**	1
🔵 **Informational (信息)**	0

2.2 Auditor's Conclusion (审计结论)

我们发现 **1 个严重漏洞，1 个高危漏洞**。该合约存在设计层面的恶意逻辑（Ponzi/Rug Pull 机制），即 `recycle` 函数导致的流动性池单向耗竭问题。由于合约所有权已放弃，这些漏洞无法被修复。**项目具有极高的崩盘风险，数学上注定会导致后入场用户的资金归零。**

---

3. Technical & Business Analysis (技术与业务分析)

3.1 Technical Quick Stats (技术概况)

主要类别	子类别	结果
合约编程	Solidity 版本	[Passed] (0.8.20)
	整数溢出/下溢	[Passed] (使用 Solidity 0.8+ 内置检查)
	函数输入参数检查	[Passed]
	访问控制管理	[Failed] (Staking 合约有未保护的资金转移函数)
	重入攻击 / 竞争条件	[Passed]
代码规范	函数可见性明确声明	[Passed]
	未使用的代码	[Passed]
Gas 优化	"Out of Gas" 问题	[Passed]
	高消耗循环	[Passed]

3.2 Business Risk Analysis (业务风险分析)

类别	结果
买卖税	[高风险] (买入2.5% LP税 + 0.5% 销毁/营销; 卖出最高 25% 利润税)
能否铸造?	[否] (固定总量)
黑名单功能?	[是] (`_rewardList` 可作为黑名单使用，但Owner已弃权无法新增)
蜜罐风险?	**[极高]** (流动性抽取机制导致后期无法卖出)
反巨鲸/机器人?	[存在缺陷] (有单笔交易限额，但可分批绕过)
隐藏 Owner?	[未检测到]
能否夺回所有权?	[否] (Owner 已设为 0x0)
审计员信心	**[低]** (代码逻辑包含恶意设计的庞氏模型)

4. Code Quality & Security (代码质量与安全)

4.1 Code Quality (代码质量)

合约使用了 OpenZeppelin 和 Solmate 等标准库，代码结构清晰，使用了 NatSpec 注释风格。然而，高质量的代码编写被用于实现恶意的经济模型逻辑。

4.2 Documentation (文档)

代码内部注释较少，缺乏对 `recycle` 等关键机制的详细业务逻辑解释，这增加了普通用户理解代码实际意图的难度。

4.3 Use of Dependencies (依赖项使用)

项目使用了 OpenZeppelin 的 IERC20 接口和 Uniswap V2 接口，以及 Solmate 的 `Owned` 权限控制。依赖项本身安全，但被调用的方式存在风险。

4.4 Centralization Risk (中心化风险)

Owner 地址已弃权（设为 `address(0)`）。虽然消除了中心化控制风险，但也意味着**目前存在的严重漏洞已无法通过升级或参数调整来修复**。合约处于永久的“僵尸”状态，按既定恶意逻辑运行。

5. Audit Findings (审计发现)

5.1 Severity Definitions (严重程度定义)

等级	描述
Critical	导致代币丢失、资产被盗或合约逻辑崩溃的漏洞。

等级	描述
High	对合约执行有重大影响或存在越权访问风险。
Medium	需要修复，但可能不会直接导致资产损失。
Low	过时代码、风格违规或轻微问题。

5.2 Detailed Findings (详细发现)

🔴 Critical (严重漏洞)

(1) 流动性池耗竭漏洞 (Liquidity Drain / Rug Pull Mechanism)Description: `Staking` 合约的 `unstake` 函数与 `LAF` 合约的 `recycle` 函数配合，构成了一个毁灭性的漏洞。
当用户解质押时：

1. 合约将 LAF 卖给 Uniswap 池子换取 USDT 支付给用户（池子 USDT 减少，LAF 增加）。
2. 合约立即调用 `LAF.recycle(amount)`。
3. `recycle` 函数强制将刚才卖入池子的 LAF 转回 `Staking` 合约。
   后果：Uniswap 流动性池中的 USDT 被单向提取，而本应留在池子中的 LAF 却被拿走了。这破坏了 AMM 的 $x*y=k$ 平衡，导致池子迅速枯竭。最终，池中将没有足够的 USDT 供后续用户兑换，合约将因滑点过大或余额不足而回滚，导致所有剩余用户的资金被永久锁定。
   Recommendation:
   此为核心逻辑缺陷。在正常开发中应移除 `recycle` 调用。但由于 Owner 已弃权，**此漏洞无法修复**。建议所有用户立即尝试解质押并撤资。
   Status: [Open / Unfixable]

🟠 High (高危漏洞)

(1) 卖出冷却时间绕过 (Cold Time Bypass)Description: `LAF` 合约在 `_transfer` 函数中限制了卖出频率，要求 `block.timestamp >= lastBuyTime[sender] + coldTime`。
然而，`lastBuyTime` 仅在用户从 Uniswap 买入时更新。用户可以通过简单的 `transfer` 操作将代币转移到一个新的钱包地址（其 `lastBuyTime` 默认为 0）。
后果：恶意用户或巨鲸可以通过多钱包转账，瞬间绕过冷却时间限制进行连续抛售，导致 `coldTime` 机制失效。
Recommendation:
应在代币转账时将发送方的 `lastBuyTime` 传递给接收方，或在转账时重置接收方的冷却时间。由于权限已弃权，**此漏洞无法修复**。
Status: [Open / Unfixable]

🟡 Medium (中危漏洞)

(1) 卖出费率逻辑的不一致性 (Profit Fee Logic Inconsistency)Description:
在 `_transfer` 的卖出逻辑中，费率计算依赖 `tOwnedU`（用户买入成本）。
如果用户将代币转移到新地址，新地址的 `tOwnedU` 为 0。代码逻辑如下：

```
} else {
    fee = amount / 4; // 25%
    tOwnedU[sender] = 0;
}
```

这意味着普通转账后的用户卖出将被收取全额 25% 的手续费，而不是基于利润的 25%。而对于原本利润极高的用户，通过转账操作可能变相降低费率或造成费率计算混乱。
Recommendation:
重新设计费率计算逻辑以适应转账场景。目前**无法修复**。
Status: [Open / Unfixable]

🟢 Low / Informational (低危/信息)

(1) 未受保护的同步函数 (Unprotected Sync Function)Description: `Staking` 合约包含一个 `sync()` 函数：

```
function sync() external {
    uint256 w_bal = IERC20(USDT).balanceOf(address(this));
    // ... transfer all USDT to pair ...
}
```

此函数为 `external` 且无权限控制。虽然 `Staking` 合约通常不持有 USDT，但在极端情况下（如有人误转 USDT 进合约），任何人都可以调用此函数将合约内的 USDT 强制捐赠给流动性池，导致资金无法找回。
Recommendation:
应添加 `onlyOwner` 修饰符。目前**无法修复**。
Status: [Open / Unfixable]

---

6. Methodology (审计方法)

Audit911 采用协作和透明的流程来提高系统质量。

1. **Manual Code Review (人工代码审查):** 我们详细检查代码逻辑、错误处理、协议解析和潜在的逻辑陷阱。
2. **Vulnerability Analysis (漏洞分析):** 我们调查威胁模型、攻击面和已知漏洞（如重入、抢跑交易、经济模型攻击）。
3. **Automated Tools (自动化工具):** 我们利用行业标准工具（如 Slither, Remix IDE）来检测常见问题。
4. **Reporting (报告):** 我们记录所有发现，验证其可行性，并基于当前合约权限状态提出修复或避险建议。
