33AUDITS & CO.

Libree Plugin Audit Report

Introduction

A time-boxed security review of the protocol was done by 33Audit & Company, focusing on the security aspects of the smart contracts. This audit was performed by 33Audits as Security Researcher.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About 33 Audits & Company

33Audits LLC is an independent smart contract security researcher company and development group. We conduct audits a as a group of independent auditors with various experience and backgrounds. We have conducted over 15 audits with dozens of vulnerabilities found and are experienced in building and auditing smart contracts. We have over 4 years of Smart Contract development with a focus in Solidity, Rust and Move. Check our previous work here or reach out on X @solidityauditor.

About Libree - 6900 Subscription Plugin

This audit is being performed on the plugin system that Libree built to add Subscription based tokens to Alchemy's ERC6900.

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Security Assessment Summary

Scope

- https://github.com/Libree/erc6900-plugins/commit/f64682a7d7565af19c076505054e03e8dbf781c7

Findings Summary

ID	Title	Severity	Status
[M-01]	Return value on transfer	Critical	Fixed
[L-01]	Fee can be bypassed	Critical	Fixed
[I-01]	USDC should have 6 decimals in tests	Critical	Fixed

[M-01] Return value of transfer not being checked

Context:

ERC20 implementations can be inconsistent. In some versions, the transfer and transferFrom functions return ‘false’ on failure instead of reverting. To handle these failures safely, it’s better to use require() statements or safe wrapper functions that check the return value/data.

Recommendations:

Replace use of IERC20(token).transfer();

This will revert on transfer failure for e.g. if msg.sender does not have a token balance >= amount.

Use the `SafeERC20` library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Resolution:

[L-01] User can bypass fee

Context: When the user calls `sendPayment` it calculates the amount the user must pay as a fee. Theoretically a user could pass in a very small amount, pay for their loan and not have to pay a fee. Though the amount would have to be extremely small and makes the impact of this act low.

```
            feeAmount = (paymentAmount * FEE) / 1e4;
            netPayment = paymentAmount - feeAmount;
```

Recommendations:

You could check that the value for `FEE` returned after calculation is greater than zero however since the impact is so small this isn't a necessary change but it was worth noting.

Resolution:

[I-01] Test cases are using 18 decimals for USDC

Context: There is no impact of this currently however just thought it should be noted that the test cases written are setting USDC decimals to 1e18 while on mainnet its 1e6. This could lead to issues in the future if you add new features.