Acala code review 2023

An incremental security audit of Acala's codebase

V1.0, September 29, 2023

Kevin Valerio kevin@srlabs.de

Jakob Rieck rieck@srlabs.de

Regina Bíró regina@srlabs.de

Abstract. This work describes the results of the thorough and independent security assurance audit for a specific set of Acala components performed by Security Research Labs. Security Research Labs is a consulting firm that has been providing specialized audit services for Substrate-based blockchains since 2019, including in the Polkadot ecosystem.

During this study, the code of in-scope components was verified to ensure that the business logic is resilient to hacking and abuse. The research team identified one medium-severity issue, one low-severity issue, and two informational severity issues.

In addition to mitigating these issues, Security Research Labs recommends adopting defensive programming principles to proactively install safeguards against unforeseen issues and enhance code robustness.

Content

1 Context

This report is intended to provide a comprehensive account of the processes, findings, and methodology employed during the security audit conducted on some specific Acala codebase components from 28th August 2023 to 29th September 2023. This audit represents an incremental assessment, building upon previous security audits performed on the same codebase, the details of which are publicly available on Acala's public GitHub repository [1].

2 Scope and methodology

The security assessment's scope, presented as a prioritized list [2], has been shared with SRLabs. This list includes various components for review, as presented in the table below.

Component name	Reference	Identified issues
Earning pallet	https://github.com/AcalaNetwork/Acala/tree/master/modules/earning	No issue was found.
LiquidCrowdloan pallet	https://github.com/AcalaNetwork/Acala/tree/master/modules/liquid-crowdloan	No issue was found.
LiquidCrowdloan precompile	https://github.com/AcalaNetwork/Acala/blob/master/runtime/common/src/precompile/liquid_crowdloan.rs	[3] [4]
Xtokens precompile	https://github.com/AcalaNetwork/Acala/blob/master/runtime/common/src/precompile/xtokens.rs	[5] [3]
StableAsset precompile	https://github.com/AcalaNetwork/Acala/blob/master/runtime/common/src/precompile/stable_asset.rs	[3]
Homa precompile	https://github.com/AcalaNetwork/Acala/blob/master/runtime/common/src/precompile/homa.rs	[3]
Acala XCM configuration	https://github.com/AcalaNetwork/Acala/blob/master/runtime/acala/src/xcm_config.rs	No issue was found.
Karura XCM configuration	https://github.com/AcalaNetwork/Acala/blob/master/runtime/karura/src/xcm_config.rs	No issue was found.

The primary focus of the security audit primarily involves manual code review. Furthermore, in addition to manual code review, the in-scope components have undergone testing using SRLabs' static and dynamic analysis tooling.

3 Findings details

The following paragraph summarizes the comprehensive findings revealed during the security audit, with the most recent update time for each status being September 29, 2023.

Issue title	Crash due to vector capacity overflow in XTokens precompile
Tracking	[5]
Severity	Medium
Status	Fixed

During the *XTokens* precompile execution, a vector allocation is performed without adequate size checks, enabling callers to provide excessively large values, which can lead to vector capacity overflows. Such overflows can trigger Rust panics, facilitating Denial-of-Service attacks. To reproduce the issue, a test input with an extremely large vector size was used, causing a panic during execution.

The risk associated with this vulnerability is significant, as attackers could potentially crash the runtime, resulting in a denial-of-service scenario. While current system contract limitations mitigate this risk, future configurations allowing user-deployed system contracts may increase exposure.

This issue has been rectified [6], ensuring that the vector allocation won’t exceed a certain threshold, configured to be safe.

Issue title	Lack of custom WeightInfo for pallet_democracy
Tracking	[7]
Severity	Low
Status	Open

Acala relies on the Substrate FRAME democracy pallet (*pallet_democracy*). The benchmarks for this pallet are conducted using the substrate-node template runtime, rather than utilizing one of Acala's specific runtimes, such as *acala*, *mandala*, or *karura*. Consequently, the *WeightInfo* defined for *pallet_democracy* in Acala's runtimes depends on benchmarked weights from the substrate-node template runtime.

This discrepancy in benchmarking environments may result in overweight or underweight extrinsics, potentially impacting the system's performance and security. To mitigate this issue, it is recommended that all pallet extrinsics, including Substrate's, should be benchmarked using the actual runtime configuration within the *define_benchmarks*! block.

Issue title	Unused context and is_static in each execute precompile function
Tracking	[3]
Severity	Info
Status	Open

Numerous precompile functions, which implement the *Precompile* trait, contain unused context and *is_static* parameters within their *execute* functions. A substantial portion of these functions do not utilize these parameters, rendering them superfluous. While this issue does not present any risk, as the required information is already defined and managed within the *transact_call* and *create_inner* functions, which handle *context* and *is_static*, it is advisable to enhance code quality by removing the redundant parameters and aligning with established best practices.

Issue title	Double gas fee charging inside LiquidCrowdloan precompiles
Tracking	[4]
Severity	Info
Status	Fixed

The *cost* function inside the *LiquidCrowdloanPrecompile* mistakenly imposed double charges of the base cost for certain actions, potentially resulting in slight overcharging for callers of the precompile. This issue has already been rectified [8], ensuring that the cost function now correctly charges only the base cost only once as intended.

4 Evolution suggestions

To fortify Acala against both known and unforeseen risks, we strongly advise adhering to the suggested remediation solutions and following the below detailed best practices.

Embrace defensive programming. Use a defensive programming approach to prevent unexpected bugs and mitigate potential security vulnerabilities, for example by using bounded vectors to limit memory abuses and utilize configurable values instead of hardcoded constants within the codebase, as is currently the case with the fee cost calculation, for example.

Conduct an economic audit. The auditors highly recommend performing an economic audit on the economic parameters of the Acala codebase, specifically examining the economic configuration items within the liquid crowdloan and earning pallets.

Address remaining security issues. Every reported security issue must be addressed as soon as possible. Even if an open issue seems to have only a minor impact, it is important to consider that malicious actors could potentially exploit it as a component in their attack strategy, potentially leading to more significant repercussions for the overall application.

Establish ecosystem best practice processes. Keep in mind the optimal procedures that are considered best practices within the Substrate ecosystem, such as performing proper benchmarking using the application's runtimes.

5 Conclusion

The primary goal of this audit is to conduct a comprehensive security assessment of various components, with a primary focus on precompiles, while also examining different pallets and XCM configurations. No critical issues were identified during this evaluation. All identified issues were promptly communicated to the Acala team through a dedicated Slack channel and have been officially documented and disclosed in our shared private GitHub repository.

6 Bibliography

[1] [Online].Available: https://github.com/AcalaNetwork/Acala/tree/master/audit.

[2] [Online]. Available: https://hackmd.io/@xlc/BJkMtRGph.

[3] [Online]. Available: https://github.com/AcalaNetwork/srlabs-audit/issues/14.

[4] [Online]. Available: https://github.com/AcalaNetwork/srlabs-audit/issues/12.

[5] [Online]. Available: https://github.com/AcalaNetwork/srlabs-audit/issues/13.

[6] [Online]. Available: https://github.com/AcalaNetwork/Acala/pull/2620.

[7] [Online]. Available: https://github.com/AcalaNetwork/srlabs-audit/issues/11.

[8] [Online]. Available: https://github.com/AcalaNetwork/Acala/pull/2611.