Security Research Labs

Acala runtime audit

Security review of the Acala runtime implementation

v1 – 22. October, 2020

Stephan Zeisberg	stephan@srlabs.de
Regina Bíró	regina@srlabs.de
Vincent Ulitzsch	vincent@srlabs.de
Jakob Lell	jakob@srlabs.de

Abstract. This report describes the results of a thorough, independent security assurance review of the Acala runtime.

SRLabs conducted a security audit on the Acala codebase - we identified 10 issues across several runtime modules, most of which were already fixed by the Laminar team.

To further improve the security of the Acala network, we suggest adhering to the benchmark-based weight annotation approach for all the runtime modules, improving documentation and leveraging continuous fuzz-testing during development.

Content

1 Scope and methodology

To provide baseline assurance about the security of the codebase of Acala, SRLabs has conducted a review on the runtime implementation of Acala, namely the runtime modules (including community maintained ORML modules that Acala depends on [1]) in scope as indicated by Laminar [2] (summarized in Table 1) and the front-end interface for the Acala blockchain [3]. The goal of this review was an in-depth analysis of the implementation of the Acala codebase. The analysis consisted of two parts:

1. Developing a threat and attack model against the Acala network (focusing on the modules in scope that could enable such attacks)
2. Reviewing and testing whether the Acala codebase is secured against all threats/attacks shown in the threat model

Repository	Runtime module	Description
ORML	Auction	Implements on-chain auctioning feature
	Authority	Implements governance features including dispatch in behalf of other accounts
	Currencies	Implements a mixed-currency system for Acala
	Gradually-update	A module for scheduling gradual updates to storage values
	Oracle	A module to allow oracle operators to feed external data
	Rewards	Implements operations with the Reward Pool
	Tokens	Provides fungible multi-currency functionality that implements the *MultiCurrency* trait
	Traits	Traits implementation for runtime modules
	Utilities	Utility implementation used across several runtime modules
	Vesting	Provides a scheduled balance lock on an account
Acala	Accounts	Responsible for opening and closing accounts on Acala
	Auction Manager	Handles auction of assets to maintain normal operation of the system
	CDP Engine	CDP engine is responsible for handling internal processes about CDPs, including liquidation, settlement and risk management
	CDP Treasury	CDP Treasury manages the accumulated interest and bad debts generated by CDPs
	DEX	Built-in decentralized exchange modules
	Emergency Shutdown	Implements the logic for triggering an emergency shutdown
	Honzon	Implements entry-points for the Honzon protocol: Allows users to manipulate their CDP position to loan/payback, and can also authorize others to manage their CDP under specific collateral type
	Incentives	A module to handle economic incentives
	Loans	Manages CDP's collateral assets and the debits backed by these assets
	Prices	Specify, lock and feed prices for and between currencies

Primitives	Define primitive type
Support	Defines support functions
Utilities	A module for opening and closing accounts in Acala, and charging fee and tip in different currencies
RenVM	Implements bridge to RenVM

Table 1 Runtime modules in scope

During the audit, we carried out a hybrid strategy utilizing a combination of code review and dynamic tests (e.g. fuzz-testing) to assess the security of the Acala codebase.

We carried out a manual code review to identify logic bugs and best practice deviations in the Acala codebase [4]. We used the master branch of the source code repositories as the basis for the review. The approach for the code review was tracing the intended functionality of the runtime modules in scope and assessing if an attacker can bypass/misuse/abuse these components or trigger unexpected behavior on the blockchain due to logic bugs or missing checks. Since the Acala codebase is entirely open-source, it is a realistic scenario that a malicious actor would start preparing for their attack by analyzing the source code.

Fuzz testing is a technique to identify issues in code which handles untrusted input, which in Acala's case is mostly the functions implementing the extrinsics (Note that the network part is handled by Substrate, which was not in scope for this review, but is built with a strong emphasis on security and where fuzz testing is also used). Fuzz testing works by taking some valid input for a method under test, applying a semi-random mutation to it, and then invoking the method under test again with this semi-valid input. Through repeating this process, fuzz-testing can unearth inputs that would cause a crash or other undefined behavior (e.g., integer overflows) in the method under test. The fuzz-testing methods we wrote utilized the test runtime Genesis configuration as well as mocked externalities to execute the fuzz test effectively against the extrinsics in scope.

Additionally as part of this review, we conducted a security audit of the acala-dapp [3] checking against common web vulnerabilities, such as Cross-site Scripting (XSS) and other injections, Cross Site Request Forgery (CSRF); as well as for the proper usage of session management, cryptographic protocols and the security of the underlying components.

2 Threat modeling and attacks

We found during our audit that the following threats can be realized given our findings against the Acala codebase:

1. **Block timeout by 'cheap' extrinsics.** A malicious actor can assemble a list of transactions that do not exceed the maximum block weight, yet still produce a timeout by including extrinsics with high resource consumption. Validator nodes will try to include this list in a block, applying the transactions in the process, but then timeout. This would cause the validator node to miss its slot and halt block production as other validators would try to put the same transactions into the block, also failing.

Missing or default weights assigned to extrinsics (described by **issue #3**) enable an attacker to realize this threat as the default weight does not reflect the computational complexity of the extrinsics.

2. **Network spamming/DoS by cheap extrinsics.** If an attacker is willing to pay the fees/high tips necessary (if any), they can include many extrinsics in a number of upcoming blocks that may delay other transactions to get included for some time. In addition, spamming could also blow up the blockchain size that would make it significantly more expensive to run an archival node with the full blockchain history. The free transaction functionality reported in **issue #11** enables an attacker to have up to 3 free transactions per day, that could be used to spam/DoS the network, especially issued with a high enough tip. Missing, 0 or default weights (**issue #3**, **issue #4**) could also enable an attacker to carry out this attack. In addition, since the transaction priority was being solely based on the fee and was independent of the length (described by **issue #9**), it enabled attackers to pay a high tip to fill up blocks with transactions of high weight or length and prevent other transactions from being included.
3. **Unfair economical advantage to (malicious) validators.** Front running in AMM DEX (Autonomous Market Maker Decentralized Exchange) is a known attack vector and is a serious issue in DeFi (Decentralized Finance) projects based on Ethereum, without any solutions available. We found during our review that **issues #5 and #8** give unfair advantages to malicious validators in the dex exchange and in bidding in auctions, respectively.

3 Findings summary

We identified 11 issues summarized in Table 2 during our analysis of the runtime modules in scope [2] in the Acala codebase that enable the above-mentioned attacks. In addition, we also reported some minor bugs and best practice deviations. In summary, one high-severity (issue #3), 7 medium severity issues and 3 low-severity/informational level issues were found. Most of the vulnerabilities were already mitigated by the Laminar team, as detailed in our findings report [5]. Acala decided to accept the risk posed by issues #5 and #8.

Issue description	Severity	Reference
Weight assignment is missing/using default values in several modules	high	#3
The *fn close_account* function is assigned 0 as weight, that allows free transactions	medium	#4
Potential for DEX market manipulation by malicious validators	medium	#5
An attacker can prevent auctions from being closed	medium	#7
Auction minimum increment issue gives advantage to validators in bidding	medium	#8
Transaction priority is based solely on the transaction fee, not taking into account its weight and length	medium	#10
Free transactions can be abused to fill up a block and artificially blow up the size of the blockchain	medium	#11
Outdated dependencies in the front-end application with known vulnerabilities	medium	#12

Bounty approval/rejection thresholds are hardcoded values	low	#2
Many of the Acala modules lack proper documentation (docstrings etc).	low	#6
In fn update_balance, taking the abs value of amount can overflow	information-level	#1

Table 2. Overview of issues identified, detailed in [5]

4 Evolution suggestion

Integrating and conducting security reviews into the development roadmap is the right path towards a security mature product. Creating a process where critical parts of the codebase (e.g. changes affecting the runtime) are reviewed thoroughly helps to harden the codebase and trains developers to develop a security mindset. By building on Substrate, Acala profits from Substrate's strong emphasis on security and Substrate's built-in protection against threats and attack affecting blockchain systems. Similarly, by choosing Rust – a memory safe programming languages – Acala has strong protection against memory safety issues such as buffer overflows or use-after-free issues.

Most of the issues that were identified by the testers have already been mitigated. However, the missing or default weight assignments for extrinsics in some runtime modules still pose a high risk, as an attacker could cause a block timeout by including lots of "cheap" extrinsics in a block whose weight does not match their computational complexity – it is strongly advised to use the benchmarked weights for all extrinsics in all the runtime modules that are considered production ready. Incomplete documentation combined with the high complexity of the Acala design and codebase increases the risk of more vulnerabilities in the future – to mitigate this risk, we recommend extending the existing documentation and code annotation, being as verbose as possible to ease further development and bug fixes. We also identified a handful of outdated components used in the Acala front-end application (detailed in [6]) – we suggest to regularly update them to a patched version as a best practice.

A review of the Acala codebase conducted by Parity [6] prior to this audit identified multiple potential integer overflows in the codebase. To make the Acala codebase more resilient and to detect this type of vulnerability early in the development lifecycle, we recommend Laminar to leverage continuous fuzz-testing. Acala can draw some inspiration from the Substrate codebase, which includes multiple fuzzing harnesses based on honggfuzz [7].

Laminar plans an audit of the economic incentives after this security audit. We strongly support this, as it will provide further assurance and uncover any misalignments in the economic incentive design.

5 References

[1] "ORML modules," [Online]. Available: https://github.com/open-web3-stack/open-runtime-module-library.

[2] "List of runtime modules in scope," [Online]. Available: https://hackmd.io/ggXkI07jQsSl2JEVl6F09g?view.

[3] "Front-end interface of Acala," [Online]. Available: https://github.com/AcalaNetwork/acala-dapp.

[4] "Acala codebase," [Online]. Available: https://github.com/AcalaNetwork.

[5] "SRL-Laminar Security Review Findings," [Online]. Available: https://securityresearchlabs.sharepoint.com/:x:/s/Laminar/EYtrsCmiKJ9EkhwAqqK80NwB-FqH8eMnPrf0HL7RxAIGjg?e=0pOdmI.

[6] "Acala-dapp outdated components," [Online]. Available: https://securityresearchlabs.sharepoint.com/sites/Laminar/Freigegebene%20Dokumente/General/3-deliverables/acala-dapp.audit.

[7] "Acala codebase review by Parity," [Online]. Available: https://github.com/AcalaNetwork/Acala/pull/356.

[8] "Honggfuzz," [Online]. Available: https://github.com/google/honggfuzz.