Acala Security Audit Report

Contents

1. Executive Summary

On Aug.11, 2020, the SlowMist security team received the Acala team's security audit application for Acala, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The Acala Dollar stablecoin (ticker: aUSD) is a multi-collateral-backed cryptocurrency, with value stable against US Dollar (aka. 1:1 aUSD to USD soft peg). It is completely decentralized, that it can be created using assets from blockchains connected to the Polkadot network including Ethereum and Bitcoin as collaterals, and can be used by any chains (or digital jurisdictions) within the Polkadot network and applications on those chains.

SlowMist blockchain system test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs suck as nodes, SDK, etc.

In black box testing and gray box testing, we use methods such as fuzz testing and script testing to test the robustness of the interface or the stability of the components by feeding random data or constructing data with a specific structure, and to mine some boundaries Abnormal performance of the system under conditions such as bugs or abnormal performance. In white box testing, we use methods such as code review, combined with the relevant experience accumulated by the security team on known blockchain security vulnerabilities, to analyze the object definition and logic implementation of the code to ensure that the code has the key components of the key logic. Realize no known vulnerabilities; at the same time, enter the vulnerability mining mode for new scenarios and new technologies, and find possible 0day errors.

SlowMist blockchain risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the blockchain, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of blockchain. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of blockchain. It is recommended to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of blockchain in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Scope of Audit

The main types of security audit include:

No.	Audit category	Subclass	Audit result
1	Code static check	RUSTSEC audit	Pass
		Numerical overflow audit	Pass
3	RPC security	Remote call permission audit	Pass
		Malformed data request audit	Pass
		Communication encryption audit	Pass
		Same-origin policy audit	Pass
4	Encrypted signature security	Random number generation algorithm audit	Pass
		Private key storage audit	Pass
		Cryptographic component call audit	Pass
		Hash intensity audit	Pass
		Transaction malleability attack audit	Pass
5	Account and transaction model security	Authority verification audit	Pass
		Transaction replay audit	Pass

		“False Top-up “ audit	Pass
6	Incentive mechanism security audit	–	Pass
7	DeFi logic security audit	–	Pass

3 Coverage

3.1 Target Code and Revision

Source	https://github.com/AcalaNetwork/Acala
Version	b0aa96c98baf16231dfe45bf81d59357e3ffcc04
Type	Blockchain base on substrate (https://github.com/paritytech/substrate)
Platform	Rust

3.2 Areas of Concern

(1). account

Module description: account opening and closing, transfer fees, etc.

Code path: modules/accounts

Audit content: Perform security scans on dependent libraries, read through codes, run unit tests, and perform security tests on account activation and closure, calculation of transfer fees, etc.

(2). auction_manager

Module description: auction, including logic for auction creation, bidding, and auction ending

Code path: modules/auction_manager

Audit content:

Perform security scans on dependent libraries, read through the code, run unit tests, sort out and understand the overall bidding process, find out the risks of the bidding process through understanding of traditional auctions, and conduct security tests on doubtful points.

(3). cdp_engine

Module description: the underlying calls within CDP, including clearing, settlement and risk management, etc.

Code path: modules/cdp_engine

Audit content: Perform security scans on dependent libraries, read through codes, run unit tests, understand processes and related methods, and conduct security tests on methods.

(4). cdp_treasury

Module description: CDP's treasury management, including interest accounting, bad debt processing through DEX and auctions, etc.

Code path: modules/cdp_treasury

Audit content: Perform security scans on dependent libraries, read through code, run unit tests,

(5). dex

Module description: Decentralized currency trading system, including the exchange of other currencies and stable currencies, the exchange between other currencies and other currencies, etc. It also provides support for the clearing of CDP

Code path: modules/dex

Audit content: Perform security scans on dependent libraries, read through the code, run unit tests, understand the underlying logic of DEX (similar to Uniswap v1), and conduct security tests on related methods and procedures.

(6). emergency_shutdown

Module description: Emergency shutdown module, when a black swan event occurs, the entire system will be shut down

Code path: modules/emergency_shutdown

Audit content: Perform security scans on dependent libraries, read through codes, run unit tests, perform security tests on related methods, and propose risk control risks based on chain understanding, and discuss feasible implementation plans with the project party.

(7). honzon

Module description:

Code path: modules/honzon

Audit content: user's method call to CDP operation, loan, repayment, etc.

Perform security scans on dependent libraries, read through the code, run unit tests, and perform security tests on related methods

(8). loans

Module description: Mortgage asset loan module

Code path: modules/loans

Audit content: Perform security scans on dependent libraries, read through codes, run unit tests, understand loan logic, and perform security tests on related methods

(9). prices

Module description: price module, stable currency pricing, other currency pricing and Oracle data acquisition, etc.

Code path: modules/prices

Audit content: Perform security scans on dependent libraries, read through the code, run unit tests, understand the dependence of price pricing, and conduct security tests on related methods.

(10). primitives

Module description: definition of some core parameters

Code path: modules/primitives

Audit content: Perform security scans on dependent libraries, read through codes, and troubleshoot security risks.

(11). support

Module description: independent definition and realization of some basic methods

Code path: modules/support

Audit content: Perform security scans on dependent libraries, read through codes, and troubleshoot security risks.

(12). utilities

Module description: some components

Code path: modules/utilities

Audit content: Perform security scans on dependent libraries, read through codes, and troubleshoot security risks.

(13). ORML

Module description: The Open Runtime Module Library (ORML) is a collection of Substrate runtime modules maintained by the community.

Code path: orml

Audit content: Perform security scans on dependent libraries, read through the code, run unit tests, understand the underlying logic of the overall system, and perform security testing in conjunction with the code implemented by modules.

4 Risk Analysis

4.1 Code static check

4.1.1 cargo audit

Automated inspection of agreed modules, check audit findings for more details.

4.1.2 Numerical overflow

Automated inspection of agreed modules, check audit findings for more details.

4.2 RPC security

4.2.1 Remote call permission audit

RPC has no sensitive permissions, and there is no "Black Valentine" vulnerability.

Reference：https://mp.weixin.qq.com/s/Kk2lsoQ1679Gda56Ec-zJg

4.2.2 Malformed data request audit

The test sent a large request, a large level of JSON, and abnormal data packets without any crash.

4.2.3 Communication encryption

Using ws for RPC interaction, non-encrypted communication will bring privacy, security and integrity risks to network participants.

4.2.4 Same-origin policy audit

Node RPC does not enable cross-domain by default.

4.3 Encrypted signature security

4.3.1 Random number generation algorithm audit

Use polkadot.js to generate the wallet private key, based on Crypto.getRandomValues(), random and unpredictable.

4.3.2 Private key storage audit

Use polkadot.js to manage the wallet's private key. The wallet has not tested the password strength, and the private key can be stored with a weak password.

4.3.3 Cryptographic component call audit

In Acala, there are four encryption layers:

Accounts

Stash Key: The Stash account is meant to hold large amounts of funds. Its private key should be as secure as possible in a cold wallet.

Controller Key: The Controller account signals choices on behalf of the Stash account, like payout preferences, but should only hold a minimal amount of funds to pay transaction fees. Its private key should be secure as it can affect validator settings, but will be used somewhat regularly for validator maintenance.

Session Keys: Session keys are "hot" keys kept in the validator client and used for signing certain validator operations. They should never hold funds.

GRANDPA: ed25519

BABE: sr25519

Online: sr25519

Parachain: sr25519

Nominator keys, nominator keys, provide a chain of trust between the stash account keys that have been mortgaged/bound and the session keys used by the node in block generation or verification. It cannot transfer DOT.

Transport layer static keys, which libp2p uses to verify the connection between nodes, use session keys.

Cryptographic component call audit

Hash function:

*sha2

*keccak

*blake2

*xxhash

Signature function:

*ed25519(Schnorr)

*sr255519(Schnorrkel)

*secp256k1

Address format:

Adopt SS58 address format

base58encode (concat (<address-type>, <address>, <checksum>))

No error calls were found.

4.3.4 Hash intensity audit

Weak hash functions such as md5 and sha1 were not found for encryption.

4.3.5 Transaction malleability attack audit

Schnorrkel signatures are unique, widely used in verifiable random functions (VRF), and have no transaction malleability.

Reference: https://github.com/w3f/schnorrkel/

4.4 Account and transaction model security

4.4.1 Authority verification audit

Substrate divides transactions into two types, one is generated by users, called extrinsic, and the other is generated by substrate itself, called inherent data.

Checked at the substrate layer, the code location is bin/node/runtime/src/lib.rs

```
fn check_extrinsics(&self, block: &$block) -> $crate::inherent::CheckInherentsResult {
use $crate::inherent::{ProvideInherent, IsFatalError};
use $crate::dispatch::IsSubType;
```

```
let mut result = $crate::inherent::CheckInherentsResult::new();
for xt in block.extrinsics() {
if $crate::inherent::Extrinsic::is_signed(xt).unwrap_or(false) {
break
}

$({
if let Some(call) = IsSubType::<_>::is_sub_type(&xt.function) {
if let Err(e) = $module::check_inherent(call, self) {
result.put_error(
$module::INHERENT_IDENTIFIER, &e
).expect("There is only one fatal error; qed");
if e.is_fatal_error() {
return result
}
}
}
})*
}

$(
match $module::is_inherent_required(self) {
Ok(Some(e)) => {
let found = block.extrinsics().iter().any(|xt| {
if $crate::inherent::Extrinsic::is_signed(xt).unwrap_or(false) {
return false
}

let call: Option<&<$module as ProvideInherent>::Call> =
xt.function.is_sub_type();

call.is_some()
});

if !found {
result.put_error(
$module::INHERENT_IDENTIFIER, &e
).expect("There is only one fatal error; qed");
if e.is_fatal_error() {
return result
}
}
},
Ok(None) => (),
Err(e) => {
result.put_error(
$module::INHERENT_IDENTIFIER, &e
).expect("There is only one fatal error; qed");
if e.is_fatal_error() {
return result
}
},
}
)*

result
}
```

4.4.2 Transaction replay audit

The fields participating in the signature include Nonce, and transactions on the same chain cannot be replayed;

- bin/node/runtime/src/lib.rs

```
pub type SignedExtra = (
frame_system::CheckSpecVersion<Runtime>,
frame_system::CheckTxVersion<Runtime>,
frame_system::CheckGenesis<Runtime>,
frame_system::CheckEra<Runtime>,
frame_system::CheckNonce<Runtime>,
frame_system::CheckWeight<Runtime>,
pallet_transaction_payment::ChargeTransactionPayment<Runtime>,
);
```

4.4.3 "False Top-up" audit

False top-up will lead to exchange accounting errors and cause capital loss.

There is no similar EOS Hard Failed mechanism.

There is no similar EVM return fail mechanism.

There is no similar USDT OP_RETURN mechanism.

There is no similar Ripple deliver_amount field.

All relevant fields of event need to be verified when recharging into the account.

4.5 Incentive mechanism security audit

By reading official documents, viewing the source code, and conducting black box testing:

https://github.com/AcalaNetwork/Acala-white-paper

https://wiki.acala.network

https://github.com/AcalaNetwork/Acala/wiki

check audit findings for more details.

4.6 DeFi logic security audit

By reading official documents, viewing the source code, and conducting black box testing:

https://github.com/AcalaNetwork/Acala-white-paper

https://wiki.acala.network

https://github.com/AcalaNetwork/Acala/wiki

check audit findings for more details.

5 Findings

Vulnerability distribution:

Critical vulnerabilities	0	
High-risk vulnerabilities	0	
Medium-risk vulnerabilities	1	🟦
Low-risk vulnerabilities	1	🟦
Weaknesses	0	
Enhancement Suggestions	2	🟩🟦
Total	4	

🟥DeFi Logic 🟥RPC Security 🟨Encrypted signature security 🟩Account and transaction model security 🟩Incentive mechanism security 🟦Code static check 🟦Other

5.1 The spin module is no longer maintained. It is recommended to replace this module.[Enhancement]

Cargo audit: 2 warnings found

```
Crate:  net2
Title:  net2 crate has been deprecated; use socket2 instead
Date:   2020-05-01
URL:    https://rustsec.org/advisories/RUSTSEC-2020-0016
------------------------------------------------------------------------------
Crate:  spin
Title:  spin is no longer actively maintained
Date:   2019-11-21
URL:    https://rustsec.org/advisories/RUSTSEC-2019-0031
```

5.2The Emergency shutdown mechanism has excessive authority and may cause adverse effects[Medium-risk]

- The modules/emergency_shutdown part is the code of the emergency shutdown mechanism. The reasons for using the function are as follows:

```
//! # Emergency Shutdown Module
//!
//! ## Overview
//!
//! When a black swan occurs such as price plunge or fatal bug, the highest
//! priority is to minimize user losses as much as possible. When the decision
//! to shutdown system is made, emergency shutdown module needs to trigger all
//! related module to halt, and start a series of operations including close
//! some user entry, freeze feed prices, run offchain worker to settle
//! CDPs has debit, cancel all active auctions module, when debits and gaps are
//! settled, the stable currency holder are allowed to refund a basket of
//! remaining collateral assets.
```

EmergencyShutdown::is_shutdown() will also be used for judgment in other modules.

The starting point of this mechanism is good, and it can be used as a means of guarantee when the chain has not undergone long-term and large-scale verification. But it also brings negative effects, such as:

a). The community sees a similar mechanism and is worried that the project party can shut down the use of the chain at any time if the authority is too large, which is not very decentralized;

b). If the private key of the project party is stolen, the impact on the chain will be too great.

Repair suggestions:

a). As far as the initial operation is concerned, the emergency shutdown mechanism can be retained, but can be enhanced:

b). The authority to call shutdown should have multiple parties to participate in the management and publicize it to the community.

c). In the long run, after the chain runs stably, this mechanism code can be removed at an opportunity.

5.3 Honzon: If the asset authorization is a complete authorization, it may cause malicious operations by the other party[Low-risk]

- modules/honzon/src/lib.rs 143-153 行

```
pub fn authorize(
origin,
currency_id: CurrencyId,
to: T::AccountId,
) {
with_transaction_result(|| {
let from = ensure_signed(origin)?;
<Authorization<T>>::insert(&from, (currency_id, &to), true);
Self::deposit_event(RawEvent::Authorization(from, to, currency_id));
Ok(())
})?;
}
```

Authorize is an asset authorization method. Users can fully authorize certain types of assets to any third-party user for processing, and user assets may be lost due to excessive authorization.

Attack scenario:

Alice authorizes account asset X to Bob, and the current account only has 100X. Alice believes that only 100X is authorized to Bob;

Transfer 10000X from Alice account;

Bob accepts Alice's 10100X as his own.

Repair suggestions:

You can refer to the previous method of Ethereum smart contract authorization, and increase the maximum authorized asset limit in the method.

5.4 Tokens transfer does not use overflow prevention methods to calculate the amount[Low-risk]

- orml/tokens/src/lib.rs 344-363

```
fn transfer(
currency_id: Self::CurrencyId,
from: &T::AccountId,
to: &T::AccountId,
amount: Self::Balance,
) -> DispatchResult {
if amount.is_zero() || from == to {
return Ok(());
}
Self::ensure_can_withdraw(currency_id, from, amount)?;

let from_balance = Self::free_balance(currency_id, from);
let to_balance = Self::free_balance(currency_id, to);
Self::set_free_balance(currency_id, from, from_balance - amount);
Self::set_free_balance(currency_id, to, to_balance + amount);
T::OnReceived::on_received(to, currency_id, amount);

Ok(())
}
```

`from_balance-amount` and `to_balance + amount` do not use overflow prevention functions for calculations.

5.5 Risk control recommendations$_{\text{[Enhancement]}}$

It is recommended to integrate the AML mechanism to ensure that the system meets compliance requirements.

6 Fix Log

(1) The Emergency shutdown mechanism has excessive authority and may cause adverse effects

-Feedback: Improvements will be made later in the project.

(2) Honzon: If the asset authorization is a complete authorization, it may cause malicious operations by the other party.

-Feedback: Substrate's low-level mechanism will not be processed temporarily.

(3) Orml/tokens transfer does not use anti-overflow method to calculate the amount.

-Feedback: The quantity is checked first in ensure_can_withdraw, and the business logic ensures that the value will not be wrong.

(4) It is recommended to integrate the AML mechanism to ensure that the system meets compliance requirements.

-Feedback: The AML mechanism is not suitable for open public chains.

7 Conclusion

Audit result: PASS

Audit No. : BCA002009070001

Audit date: November 7, 2020

Audit team: SlowMist security team

Summary conclusion: After correction, all problems found have been fixed and the above risks have been eliminated by Acala. Comprehensive assessed, Acala has no risks above already.

8. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.
For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.