Security Assessment Report

Lulo

May 02, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Lulo smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 8 issues or questions.

program	type	commit
Lulo	Solana	`781a0d3d7640207c324253bc15292abb690ea22d`

Per team's instruction, the following 3 instructions were excluded from this review:

- `"init_action"`
- `"claim_tokens"`
- `"fund_tokens"`

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
LULO		
[M-01] Missing obligation and promotion_authority consistency check	Medium	Resolved
[M-02] Arbitrary CPIs	Medium	Resolved
[L-01] Outdated liquidity_token_account amount	Low	Resolved
[L-02] Potential DoS issue caused by the init attribute of the ATA accounts	Low	Resolved
[I-01] Failure when closing liquidity_token_account	Info	Resolved
[I-02] Inconsistent macro naming in WithdrawMarginFi	Info	Resolved
[I-03] Runtime overflow check not enabled	Info	Resolved
[I-04] Duplicated code snippets	Info	Resolved

Findings in Detail

LULO

[M-01] Missing obligation and promotion_authority consistency check

In the current implementation of "`deposit/withdraw_2x_kamino`", the accounting logic operates as follows:

- In deposit, the "`promotion_authority.total_deposits`" records the liquidity token amount;
- In withdrawal, after "`WithdrawObligationCollateralAndRedeemReserveCollateral`", the tokens obtained are proportionally returned to users based on the corresponding amounts recorded, with any surplus being returned to the reserve.

However, this design works for a one-to-one correspondence between a "`promotion_authority`" and a liquidity mint, neglecting the possibility that multiple lending markets or obligations could share the same "`promotion_authority`".

Consider a scenario where a malicious user deposits 100 tokens into obligation A and another 100 tokens into obligation B, then proceeds to withdraw from obligation A.

In this case, the attacker could retrieve all 200 tokens, including the 100 tokens from the admin reserve, while the 200 tokens that should have been returned to the admin reserve remain in obligation B. By repeating such actions, the attacker could potentially execute a DoS attack.

Although administrators may detect such malicious behavior and manually reclaim tokens through transfer authority, it is recommended to introduce checks to ensure a one-to-one correspondence between a "`promotion_authority`" and its obligation.

Resolution

This issue has been resolved by commit `1fcd51ea596878ecb31a78363da075d2c9b343da`.

LULO

[M-02] Arbitrary CPIs

In the Solend-related CPIs, the Solend program address ("solend_program") is not validated. As a result, an attacker could use a malicious program address as "solend_program" and call a crafted smart contract with the signature of the "user_account" PDA.

For instance, in the "deposit_solend" and "withdraw_solend", the "collateral_token_account" and "liquidity_token_account" token accounts' authority is the signer of the CPI calls. If attackers use the program ID of a malicious smart contract as the "solend_program" and deceive users into signing the transaction, they could potentially steal the tokens held in the aforementioned token accounts inside this maliciously crafted CPI callee program, or just burn the tokens.

It is recommended to validate the "solend_program" program ID to ensure security.

Furthermore, while it is currently considered safe to omit program ID checks in the CPI calls to "mfi", "drift", "kamino", and "mango", due to the CPI program ID in Anchor version 0.28 being hardcoded as "crate::ID" (see cpi.rs in anchor 0.28). This approach needs reevaluation with the release of Anchor 0.29. In this newer version, the CPI program ID is set to "ctx.program.key()" (see cpi.rs in anchor 0.29).

Consequently, when upgrading to Anchor 0.29, validating the program ID in CPI calls becomes necessary, as these IDs now originate from unchecked accounts in the context.

1. init_solend

```
/* programs/flexlend/src/solend.rs */
008 | pub struct InitSolend<'info> {
036 |     #[account()]
037 |     /// CHECK: CPI
038 |     pub solend_program: AccountInfo<'info>,

045 | pub fn init_solend(ctx: Context<InitSolend>) -> Result<()> {
056 |     let instruction = init_obligation(
057 |         ctx.accounts.solend_program.key(),
063 |     );
064 |     invoke_signed(
065 |         &instruction,
075 |     )?;
```

```
379 | pub fn init_obligation(
380 |     program_id: Pubkey,
386 | ) -> Instruction {
387 |     Instruction {
388 |         program_id,
```

2. deposit_solend

```
/* programs/flexlend/src/solend.rs */
085 | pub struct ConvertSolend<'info> {
164 |     #[account()]
165 |     /// CHECK: CPI
166 |     pub solend_program: AccountInfo<'info>,

179 | pub fn deposit_solend(
180 |     ctx: Context<ConvertSolend>,
184 | ) -> Result<()> {
195 |     let solend_ix = deposit_reserve_liquidity(
196 |         ctx.accounts.solend_program.key(),
207 |         ctx.accounts.user_account.key(), // obligation_owner
210 |         ctx.accounts.user_account.key(), // user_transfer_authority_pubkey

238 | fn deposit_reserve_liquidity(
239 |     program_id: Pubkey,
250 |     obligation_owner: Pubkey,
253 |     user_transfer_authority_pubkey: Pubkey,
255 | ) -> Instruction {
260 |     Instruction {
261 |         program_id,
```

3. withdraw_solend

```
/* programs/flexlend/src/solend.rs */
085 | pub struct ConvertSolend<'info> {
164 |     #[account()]
165 |     /// CHECK: CPI
166 |     pub solend_program: AccountInfo<'info>,

282 | pub fn withdraw_solend(
283 |     ctx: Context<ConvertSolend>,
287 | ) -> Result<()> {
294 |     let solend_ix = redeem_reserve_collateral(
295 |        ctx.accounts.solend_program.key(),
306 |        ctx.accounts.user_account.key(), // obligation_owner
307 |        ctx.accounts.user_account.key(), // user_transfer_authority_pubkey

335 | fn redeem_reserve_collateral(
336 |     program_id: Pubkey,
351 |     obligation_owner: Pubkey,
352 |     user_transfer_authority_pubkey: Pubkey, // was 7, now 10
```

```
354 | ) -> Instruction {
359 |     Instruction {
360 |         program_id,
```

Resolution

This issue has been resolved by commit `f94f0850400678cd43e0ea969e4d32b26fba6cc4`.

LULO

[L-01] Outdated liquidity_token_account amount

The balance of "`liquidity_token_account`" changes after the "`transfer`" operation at line 222 in "`kamino.rs`".

In line 244, "`ctx.accounts.liquidity_token_account`" points to a local copy loaded before the transfer, resulting in "`liquidity_amount`" being assigned by an outdated value.

A "`reload()`" operation is necessary after the transfer to update this value.

```
/* programs/flexlend/src/kamino.rs */
208 | pub fn deposit_kamino(
209 |     ctx: Context<DepositKamino>,
210 |     deposit_amount: u64,
211 |     deposit_all: bool,
212 |     is_migration: bool,
213 | ) -> Result<()> {
220 |     if automation.key().eq(&user.owner) {
221 |         // Transfer from user to liquidity token account
222 |         anchor_spl::token::transfer(
223 |             CpiContext::new(
224 |                 ctx.accounts.token_program.to_account_info(),
225 |                 anchor_spl::token::Transfer {
226 |                     from: ctx.accounts.user_token_account.to_account_info(),
227 |                     to: ctx.accounts.liquidity_token_account.to_account_info(),
228 |                     authority: ctx.accounts.owner.to_account_info(),
229 |                 },
230 |             ),
231 |             deposit_amount,
232 |         )?;
233 |     }
243 |     let liquidity_amount = if is_migration || deposit_all {
244 |         ctx.accounts.liquidity_token_account.amount
245 |     } else {
246 |         deposit_amount
247 |     };
```

Resolution

This issue has been resolved by commit fa53c833ab800dcaeb8eb0468d7f53ea7cf6f079.

LULO

[L-02] Potential DoS issue caused by the init attribute of the ATA accounts

In "Deposit2xKamino" and "Withdraw2xKamino", the "liquidity_token_account" has an "init" attribute, requiring it to be an uninitialized account.

However, since "liquidity_token_account" is an associated token account (ATA), it could have been created by anyone before. If such an account is pre-created, it would prevent users from normally depositing or withdrawing, potentially leading to a denial of service (DoS).

Consider changing the "init" attribute on "liquidity_token_account" to "init_if_needed" to mitigate this risk.

```
/* programs/flexlend/src/d2x/instructions/deposit_2x_kamino.rs */
014 | pub struct Deposit2xKamino<'info> {
079 |     #[account(
080 |         init,
081 |         payer = owner,
082 |         associated_token::mint = liquidity_mint_address,
083 |         associated_token::authority = promotion_authority,
084 |     )]
085 |     pub liquidity_token_account: Box<Account<'info, TokenAccount>>,

/* programs/flexlend/src/d2x/instructions/withdraw_2x_kamino.rs */
014 | pub struct Withdraw2xKamino<'info> {
080 |     #[account(
081 |         init,
082 |         payer = owner,
083 |         associated_token::mint = liquidity_mint_address,
084 |         associated_token::authority = promotion_authority,
085 |     )]
086 |     pub liquidity_token_account: Box<Account<'info, TokenAccount>>,
```

Resolution

This issue has been resolved by commit `eeb08f37cdf28f29eee774329d5ad446e087955b`.

LULO

[I-01] Failure when closing liquidity_token_account

In the "deposit_kamino" function, closing the "liquidity_token_account" may fail if there is a remaining balance in the account.

```
/* programs/flexlend/src/kamino.rs */
208 | pub fn deposit_kamino(
213 | ) -> Result<()> {
243 |     let liquidity_amount = if is_migration || deposit_all {
244 |         ctx.accounts.liquidity_token_account.amount
245 |     } else {
246 |         deposit_amount
247 |     };
248 |
249 |     msg!("liquidity_amount: {}", liquidity_amount);
250 |
251 |     kamino_cpi::cpi::deposit_reserve_liquidity_and_obligation_collateral(
252 |         CpiContext::new_with_signer(
278 |         ),
279 |         liquidity_amount,
280 |     )?;
281 |
282 |     if automation.key().eq(&user.owner) {
283 |         // Close liquidity_token_account
284 |         anchor_spl::token::close_account(CpiContext::new_with_signer(
285 |             ctx.accounts.token_program.to_account_info(),
286 |             anchor_spl::token::CloseAccount {
287 |                 account: ctx.accounts.liquidity_token_account.to_account_info(),
288 |                 destination: ctx.accounts.owner.to_account_info(),
289 |                 authority: ctx.accounts.user_account.to_account_info(),
290 |             },
291 |             signer_seeds,
292 |         ))?;
293 |     }
```

```
/* spl-token-3.5.0/src/processor.rs */
664 | /// Processes a [CloseAccount](enum.TokenInstruction.html) instruction.
665 | pub fn process_close_account(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult {
675 |     let source_account = Account::unpack(&source_account_info.data.borrow())?;
676 |     if !source_account.is_native() && source_account.amount != 0 {
677 |         return Err(TokenError::NonNativeHasBalance.into());
678 |     }
```

Consider transferring the balance from "liquidity_token_account" to "user_token_account" in advance.

Resolution

This issue has been resolved by commit `fa53c833ab800dcaeb8eb0468d7f53ea7cf6f079`.

LULO

[I-02] Inconsistent macro naming in WithdrawMarginFi

The "`withdraw_marginfi`" function uses parameters prefixed with "`withdrawal_*`". However, the parameters in "`WithdrawMarginFi`" are prefixed with "`deposit_*`".

This isn't an issue per se, but aligning these prefixes could enhance readability.

```
/* programs/flexlend/src/lib.rs */
107 | pub fn withdraw_marginfi<'a, 'b, 'c, 'info>(
108 |     ctx: Context<'a, 'b, 'c, 'info, WithdrawMarginFi<'info>>,
109 |     withdrawal_amount: u64,
110 |     withdraw_all: bool,
111 |     is_migration: bool,
112 | ) -> Result<()> {
113 |     mfi::withdraw_marginfi(ctx, withdrawal_amount, withdraw_all, is_migration)
114 | }

/* programs/flexlend/src/mfi.rs */
128 | #[derive(Accounts)]
129 | #[instruction(deposit_amount: u64, deposit_all: bool, is_migration: bool)]
130 | pub struct WithdrawMarginFi<'info> {
```

Resolution

This issue has been resolved by commit 6473da0c6200a692a2b93dd1651573c9e563ab68.

LULO

[I-03] Runtime overflow check not enabled

It is generally a good practice to activate runtime overflow checks for the following two arithmetic operations, even though an overflow may not occur.

Specifically, in "`deposit_2x_kamino.rs:188`", the "`amount`" is a "`u64`" variable, so "`amount * 2`" could potentially overflow. However, this is unlikely to happen because the two preceding transfer operations to the "`liquidity_token_account`" would fail first.

```
/* flexlend/src/d2x/instructions/deposit_2x_kamino.rs */
123 | pub fn deposit_2x_kamino(ctx: Context<Deposit2xKamino>, amount: u64) -> Result<()> {
129 |     // Transfer from reserve to liquidity token account
130 |     anchor_spl::token::transfer(
131 |         CpiContext::new_with_signer(
133 |             anchor_spl::token::Transfer {
135 |                 to: ctx.accounts.liquidity_token_account.to_account_info(),
137 |             },
139 |         ),
140 |         amount,
141 |     )?;
142 |
143 |     // Transfer from user to liquidity token account
144 |     anchor_spl::token::transfer(
145 |         CpiContext::new(
147 |             anchor_spl::token::Transfer {
149 |                 to: ctx.accounts.liquidity_token_account.to_account_info(),
151 |             },
152 |         ),
153 |         amount,
154 |     )?;
159 |     // Deposit 2x
160 |     kamino_cpi::cpi::deposit_reserve_liquidity_and_obligation_collateral(
161 |         CpiContext::new_with_signer(
187 |         ),
188 |         amount * 2,
189 |     )?;
```

Additionally, when calculating "`reclaim`", although "`promo_info.lamports() - minimum_rent`" won't underflow because the "`2115840`" is the calculated rent for the "`promo_info`" account, it is still good practice to enable the runtime overflow check.

```
/* flexlend/src/d2x/instructions/withdraw_2x_kamino.rs */
125 | pub fn withdraw_2x_kamino(
126 |     ctx: Context<Withdraw2xKamino>,
127 |     _amount: u64,
```

```
128 |     _withdraw_all: bool,
129 | ) -> Result<()> {
220 |     let minimum_rent: u64 = 2115840;
222 |     let promo_info = ctx.accounts.promotion_authority.to_account_info();
223 |     let wallet_info = ctx.accounts.owner.to_account_info();
225 |     let reclaim = promo_info.lamports() - minimum_rent;
227 |     if reclaim > 0 {
228 |         **promo_info.try_borrow_mut_lamports()? -= reclaim;
229 |         **wallet_info.try_borrow_mut_lamports()? += reclaim;
230 |     }
```

Resolution

This issue has been resolved by commit `d62fb926477ee024113c0e3b0093eb76069318dc`.

LULO

[I-04] Duplicated code snippets

The statement suggests that two sections of code (lines 279-284 and lines 286-291) are the same, indicating redundancy. To optimize, consider merging them into a single section or function.

```
/* programs/flexlend/src/mfi.rs */
252 | pub fn withdraw_marginfi<'a, 'b, 'c, 'info>(
257 | ) -> Result<()> {
278 |     let cpi_ctx = if withdraw_all {
279 |         anchor_lang::prelude::CpiContext {
280 |             program: ctx.accounts.mfi_program.to_account_info(),
281 |             accounts: accts,
282 |             remaining_accounts: ctx.remaining_accounts.to_vec(),
283 |             signer_seeds,
284 |         }
285 |     } else {
286 |         anchor_lang::prelude::CpiContext {
287 |             program: ctx.accounts.mfi_program.to_account_info(),
288 |             accounts: accts,
289 |             remaining_accounts: ctx.remaining_accounts.to_vec(),
290 |             signer_seeds,
291 |         }
292 |     };
```

Resolution

This issue has been resolved by commit `3d305f33122ba8bc61a2d1154c345207c7cf71dc`.

Appendix: Methodology and Scope of Work

The Sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the Sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Lulo Labs Inc. (the "Client''). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

Sec3™