Security Assessment Report

Unstake Smart Contract

July 24th, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Unstake Solana Smart Contract. The artifact of the audit was the source code of the following smart contracts excluding tests in a private repository.

- programs/unstake

The initial audit was done on commit e2e87ce9511bba60f21b0c9e0cb1bb1086b6d979 of the following smart contracts and shared utilities.

The audit revealed 2 issues or questions. This report describes the findings and resolutions in detail.

Table of Contents

Result Overview

Issue	Impact	Status
[C-1] LP token price manipulation caused by flash loan	Critical	Resolved
[I-1] Referrer's account is not checked	Informational	Resolved

Findings in Detail

[C-1] LP token price manipulation caused by flash loan

Take a flash loan:

```
/* programs/unstake/src/instructions/flash_loan/take_flash_loan.rs */
132 | // transfer to receiver
133 | transfer(
134 |     CpiContext::new_with_signer(
135 |         system_program.to_account_info(),
136 |         Transfer {
137 |             from: pool_sol_reserves.to_account_info(),
138 |             to: receiver.to_account_info(),
139 |         },
140 |         &[seeds],
141 |     ),
142 |     lamports,
143 | )?;
```

Repay the flash loan:

```
/* programs/unstake/src/instructions/flash_loan/repay_flash_loan.rs */
090 | transfer(
091 |     CpiContext::new(
092 |         system_program.to_account_info(),
093 |         Transfer {
094 |             from: repayer.to_account_info(),
095 |             to: pool_sol_reserves.to_account_info(),
096 |         },
097 |     ),
098 |     repay_lamports,
099 | )?;
```

When borrowing a flash loan, the source of funds is `pool_sol_reserves`, When repaying the flash loan, the funds are directly transferred to `pool_sol_reserves`.

However, the quantity of lamports in `pool_sol_reserves` directly affects the calculation of the LP token price, which creates the possibility of price manipulation attacks in `remove_liquidity`.

```
/* programs/unstake/src/instructions/remove_liquidity.rs */
058 | // order matters, must calculate first before mutation
059 | let pool_owned_lamports = pool_sol_reserves
060 |     .lamports()
061 |     .checked_add(pool_account.incoming_stake)
062 |     .ok_or(UnstakeError::InternalError)?;
063 | let to_return = calc_lamports_to_return(pool_owned_lamports, lp_mint.supply, amount_lp)?;

099 | fn calc_lamports_to_return(
100 |     pool_owned_lamports: u64,
101 |     lp_mint_supply: u64,
102 |     amount_lp_to_burn: u64,
103 | ) -> std::result::Result<u64, UnstakeError> {
104 |     // 0 edge-cases: return 0
105 |     if pool_owned_lamports == 0 || lp_mint_supply == 0 {
106 |         return Ok(0);
107 |     }
108 |     // return = amount_lp_to_burn * owned_lamports BEFORE BURN / lp_mint.supply BEFORE BURN
109 |     u128::from(amount_lp_to_burn)
110 |         .checked_mul(u128::from(pool_owned_lamports))
111 |         .and_then(|v| v.checked_div(u128::from(lp_mint_supply)))
112 |         .and_then(|v| u64::try_from(v).ok())
113 |         .ok_or(UnstakeError::InternalError)
114 | }
```

Consider the following attack scenario for a pool with 1000 lamports and 1000 LP tokens minted. For simplicity, let's ignore the fees.

- Instruction 1: An attacker borrows 999 lamports.
- Instruction 2: The attacker adds 1000 lamports to the pool's liquidity, which will mint them `1000 * 1000 / 1=1000000` LP tokens.
- Instruction 3: The attacker repays the flash loan of 999 lamports.
- Instruction 4: The attacker reduces the liquidity and burns their 1000000 LP tokens to redeem lamports. The number of lamports they can obtain is calculated as `1000000 * 2000 / 1001000 = 1998` lamports, almost emptying the pool.

Resolution

The borrowed amount is treated as the funds in the pool. This issue is resolved.

[I-1] Referrer's account is not checked

```
/* programs/unstake/src/instructions/unstake_instructions/unstake_accounts.rs */
162 | let lamports_to_protocol = match Self::referrer(ctx) {
164 |     Some(referrer) => {
165 |         let lamports_to_referrer = ctx
166 |             .accounts
167 |             .protocol_fee_account()
168 |             .apply_referrer_fee(protocol_fee_lamports)
170 |         let lamports_to_protocol = protocol_fee_lamports
171 |             .checked_sub(lamports_to_referrer)
172 |             .ok_or(UnstakeError::InternalError)?;
174 |         // pay the referrer fees from the pool reserves
175 |         let referrer_fee_transfer_cpi_accs = system_program::Transfer {
176 |             from: ctx.accounts.pool_sol_reserves().to_account_info(),
177 |             to: referrer,
178 |         };
179 |         system_program::transfer(
180 |             CpiContext::new_with_signer(
181 |                 ctx.accounts.system_program().to_account_info(),
182 |                 referrer_fee_transfer_cpi_accs,
183 |                 &[pool_sol_reserves_seeds],
184 |             ),
185 |             lamports_to_referrer,
186 |         )?;
189 |     }
190 | };
```

When unstaking or repaying a flash loan, if the referrer is provided, a portion of the protocol fee will be transferred to the referrer's account. However, there are currently no checks in place on the referrer's account, which could potentially allow users to partially avoid paying a small portion of the protocol fee.

Resolution

The team acknowledged the finding. However, determining if the referrer account is controlled by the user is not straightforward because the user can always create a new wallet and use it as the referrer.

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 Solana 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.
    - Missing ownership checks
    - Missing signer checks
    - Signed invocation of unverified programs
    - Solana account confusions
    - Arithmetic over- or underflows
    - Numerical precision errors
    - Loss of precision in calculation
    - Insufficient SPL-Token account verification
    - Missing rent exemption assertion
    - Casting truncation
    - Did not follow security best practices
    - Outdated dependencies
    - Redundant code
    - Unsafe Rust code
- 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 Socean Foundation (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.