HALBORN

DRAFT

# StaderLabs - Liquid Staking

## Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 12th, 2022 - March 21st, 2022**

Visit: **Halborn.com**

# DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE | AUTHOR |
|---|---|---|---|
| 0.1 | Document Creation | 02/14/2022 | Isabel Burruezo |
| 0.2 | Draft Review | 02/24/2022 | Gabi Urrutia |

# CONTACTS

| CONTACT | COMPANY | EMAIL |
|---|---|---|
| Rob Behnke | Halborn | Rob.Behnke@halborn.com |
| Steven Walbroehl | Halborn | Steven.Walbroehl@halborn.com |
| Gabi Urrutia | Halborn | Gabi.Urrutia@halborn.com |
| Isabel Burruezo | Halborn | Isabel.Burruezo@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

Stader Labs engaged Halborn to conduct a security audit on their smart contracts beginning on February 12th and ending on March 6th. The security assessment was scoped to the smart contracts provided to the Halborn team.

## 1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that should be addressed by the team.

## 1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Fuzz testing. (Halborn custom fuzzing tool).
- Checking the test coverage. (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities.(cargo audit).

https://github.com/stader-labs/solana-pool-liquid-staking/commit/c8dc412f093ec19539

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

**RISK SCALE - LIKELIHOOD**

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

**RISK SCALE - IMPACT**

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.


**10** - CRITICAL
**9 - 8** - HIGH
**7 - 6** - MEDIUM
**5 - 4** - LOW
**3 - 1** - VERY LOW AND INFORMATIONAL

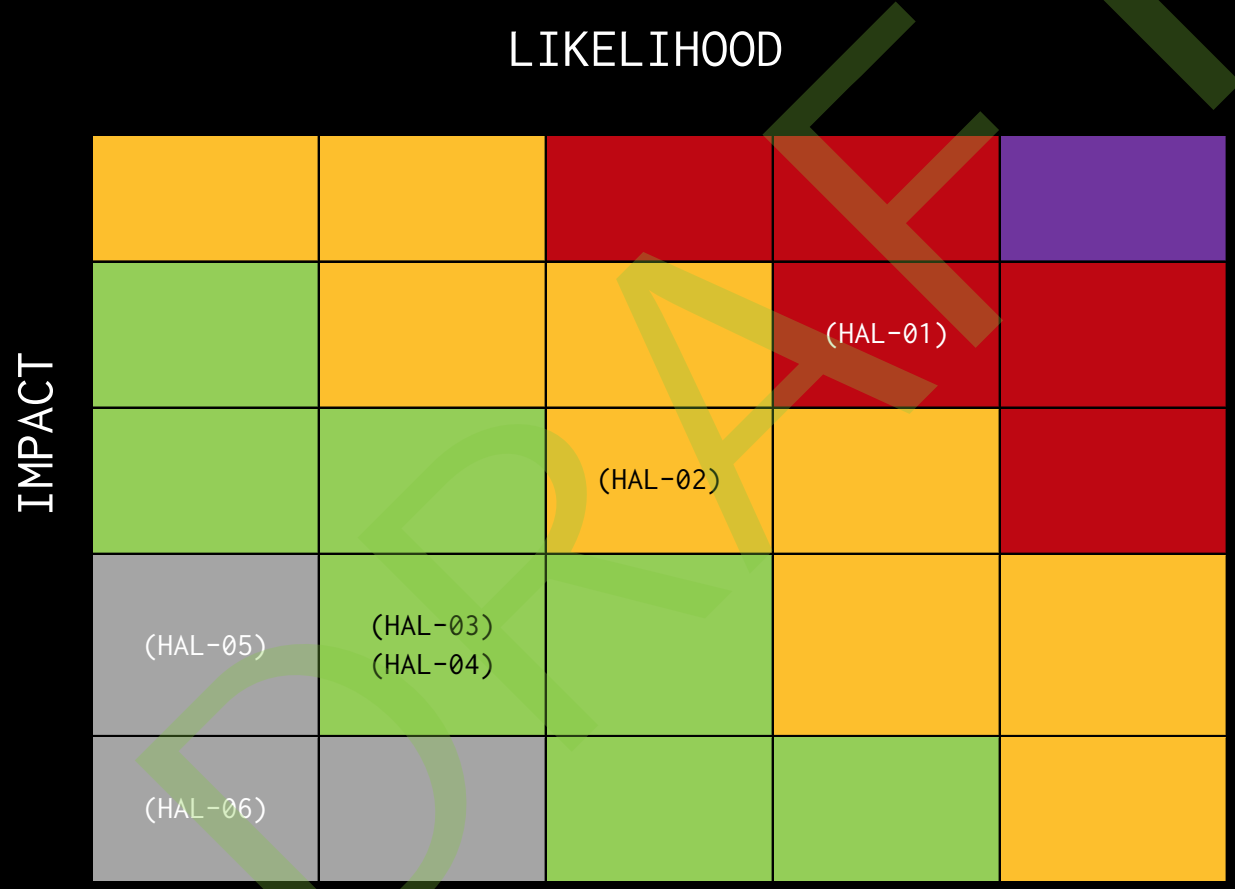
## 1.4 SCOPE

This review was scoped to the Solana Program Audit Branch.

1. Solana program
   (a) Repository: solana-pool-liquid-staking
   (b) Commit ID: c8dc412f093ec195392149513c7590cc54c02830

# 2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 |



| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| (HAL-01) POOL TOKEN FEES BURNT AND NOT TRANSFERABLE | High | - |
| (HAL-02) MANAGER ADDRESS CANNOT BE TRANSFERRED | Medium | - |
| (HAL-03) FEES VERIFICATION MISSING | Low | - |
| (HAL-04) USE OF UNSAFE CODE IN AN UNNECESSARY FUNCTION | Low | - |
| (HAL-05) PREFERRED WITHDRAW ACCOUNT MISSING | Informational | - |
| (HAL-06) SOME INCORRECT TRACE MESSAGE | Informational | - |

# FINDINGS & TECH DETAILS

## 3.1 (HAL-01) POOL TOKEN FEES BURNT AND NOT TRANSFERABLE - HIGH

Description:

Users have to pay some operating fees to the staking pool. On pool initialization, the pool fee token account was created, and when initialized, its authority is set to spl_token program instead of manager. This means that when users pay fees, some tokens are minted to that pool fee account, but they are not transferable because the manager cannot sign because it is not the owner so, the fees are simply burned.

Code Location:

**Listing 1: processor/initialize.rs (Line 212)**

```
207  invoke_signed(
208          &spl_token::instruction::initialize_account(
209              &accounts.spl_token_program_id.key,
210              &accounts.pool_fee.key,
211              &accounts.pool_mint.key,
212              &accounts.spl_token_program_id.key,
213          )?,
214          &[
215              accounts.sysvar_rent_id.clone(),
216              accounts.pool_mint.clone(),
217              accounts.pool_fee.clone(),
218          ],
219          &[&[
220              &accounts.stader.key.to_bytes(),
221              pda::pool_mint::SEED,
222              &[bump_seeds.pool_mint],
223          ]],
224      )?;
225      crate::trace!("Initialized: spl_token Account.");
```

Risk Level:

**Likelihood - 4**
**Impact - 4**

Recommendation:

The `pool_fee` token account authority must be set to the manager during initialization.

## 3.2 (HAL-02) MANAGER ADDRESS CANNOT BE TRANSFERRED - MEDIUM

Description:

The program lacks the option to set a new manager as a privileged address. If the development team needed to change the manager address for an operational reason, a significant portion of the contract's functionality would be unusable.

Code Location:

**Solana-pool-liquid-staking program**

Risk Level:

**Likelihood - 3**
**Impact - 3**

Recommendation:

It is recommended to add **manager transfer** capabilities to the program.

## 3.3 (HAL-03) FEES VERIFICATION MISSING - LOW

Description:

By design, it may be desirable to set some fees to zero and others to non-zero. However, those that are not desired to be zero should be verified.
In addition, outside of monetization, fees are a crucial tool to prevent economic attacks on the stake pool and keep it running.

Code Location:

**Listing 2: processor/set_fees.rs**

```
23  Stader::check_is_rent_exempt(
24          &rent::Rent::from_account_info(accounts.sysvar_rent_id)?,
25          accounts.stader,
26      )?;
27
28      let stader = Stader::check_deserialize(program_id, accounts.
            stader)?.state()?;
29
30      // Check the program derived accounts
31      pda::check_is_valid(
32          pda::stake_pool::create(
33              program_id,
34              accounts.stader.key,
35              stader.bump_seeds.stake_pool,
36          ),
37          accounts.stake_pool,
38      )?;
39
40      match epoch_fee {
41          None => {
42              crate::trace!("epoch_fee update not required.");
43          }
44          Some(fee) => {
45              invoke(
46                  &spl_stake_pool::instruction::set_fee(
```

```
47                      accounts.spl_stake_pool_program_id.key,
48                      accounts.stake_pool.key,
49                      &accounts.manager.key,
50                      FeeType::Epoch(fee),
51                  ),
52                  account_infos,
53              )?;
54              crate::trace!("Updated epoch_fee to {}", fee);
55          }
56      }
57
```

Risk Level:

**Likelihood - 2**
**Impact - 2**

Recommendations:

It is recommended to verify that the fees to be earned are different from zero.

## 3.4 (HAL-04) USE OF UNSAFE CODE IN AN UNNECESSARY FUNCTION - LOW

Description:

Although the Rust programming language is memory-safe by default, it allows the user to provide the unsafe keyword/feature to apply fewer restrictions than normal. Use of unsafe code is possible to dereference a raw pointer, read or write a mutable or external static variable, access a field of a union other than to assign to it, call an unsafe function or implement an unsafe trait. The security consequences of using unsafe code in Rust increases the chances of being exposed to various vulnerabilities or bugs, leading to memory leaks. Worst cases can expose sensitive information left in memory, or gain remote code execution by taking control of the pointer in memory, and redirecting it to attacker-controlled malicious code execution sectors.

In addition, this unsafe code is placed in an unnecessary function since the owner of the payer is always the manager or staker, so the owner is the system program.

Code Location:

**Listing 3: processor/utils.rs (Lines 67,68)**

```
57 /// Creates an account with rent exemption,
58 /// even if the system does not own the payer.
59 pub fn create_account_no_ownership<'a>(
60     rent: &rent_program::Rent,
61     space: usize,
62     account: &AccountInfo<'a>,
63     owner: &Pubkey,
64     payer: &AccountInfo<'a>,
65     signers_seeds: &[&[&[u8]]],
66 ) -> ProgramResult {
67     let set_payer_owner_to = |to: &Pubkey| unsafe {
68         std::ptr::write_volatile(payer.owner as *const Pubkey as *
            mut [u8; 32], to.to_bytes());
```

```
69      };
70
71      let payer_owner = payer.owner;
72
73      // Set the system as the payer owner, so the create account
            can be called correctly:
74      set_payer_owner_to(&solana_program::system_program::id());
75
76      let amount = rent.minimum_balance(space) + TRANSACTION_FEE;
77      let space = space as u64;
78      invoke_signed(
79          &system_instruction::create_account(&payer.key, &account.
                key, amount, space, owner),
80          &[payer.clone(), account.clone()],
81          signers_seeds,
82      )?;
83
84      // Set the payer owner back to its original value:
85      set_payer_owner_to(payer_owner);
86
87      Ok(())
88  }
```

Risk Level:

**Likelihood - 2**
**Impact - 2**

Recommendation:

It is recommended not to use unsafe code to avoid exposing possible vulnerabilities or bugs that trigger memory leaks, or to remove this function.

## 3.5 (HAL-05) PREFERRED WITHDRAW ACCOUNT MISSING - INFORMATIONAL

Description:

The staker can set a preferred withdrawal account, which forces users to withdraw funds from a particular staking account. This prevents malicious depositors from using the stake pool as a free conversion between validators.

Code Location:

**Solana-pool-liquid-staking program**

Risk Level:

**Likelihood - 1**
**Impact - 2**

Recommendation:

Setting a “preferred withdraw account” is recommended.

## 3.6 (HAL-06) SOME INCORRECT TRACE MESSAGES - INFORMATIONAL

Description:

There are trace messages in the decrease_stake and remove_validator instruction handlers that do not correspond to what is being done in it.

Code Location:

**Listing 4: processor/remove_validator.rs (Line 114)**

```
87 invoke(
88         &spl_stake_pool::instruction::
               remove_validator_from_pool_with_vote(
89             accounts.spl_stake_pool_program_id.key,
90             &stake_pool,
91             accounts.stake_pool.key,
92             accounts.validator.key,
93             accounts.spp_withdraw_authority.key, //
                   new_stake_account_authority
94             spp_transient_stake_seed,
95             accounts.transient_deactivating_stake.key, // the
                   split stake account (transient one)
96         ),
97         &[
98             accounts.sysvar_rent_id.clone(),
99             accounts.sysvar_clock_id.clone(),
100            accounts.sysvar_stake_history_id.clone(),
101            accounts.sysvar_stake_id.clone(),
102            accounts.sysvar_stake_config_id.clone(),
103            accounts.spp_validator_stake.clone(),
104            accounts.spp_withdraw_authority.clone(),
105            accounts.spp_transient_stake.clone(),
106            accounts.stake_pool.clone(),
107            accounts.staker.clone(),
108            accounts.validators_list.clone(),
109            accounts.validator.clone(),
110            accounts.transient_deactivating_stake.clone(),
111        ],
112    )?;
```

```
113
114     crate::trace!("Validators added");
115
```

**Listing 5: processor/decrease_validator.rs (Line 73)**

```
48      invoke(
49          &spl_stake_pool::instruction::
                decrease_validator_stake_with_vote(
50              accounts.spl_stake_pool_program_id.key,
51              &stake_pool,
52              accounts.stake_pool.key,
53              accounts.validator.key,
54              lamports,
55              spp_transient_stake_seed,
56          ),
57          &[
58              accounts.staker.clone(),
59              accounts.stake_pool.clone(),
60              accounts.validators_list.clone(),
61              accounts.reserve_stake.clone(),
62              accounts.spp_withdraw_authority.clone(),
63              accounts.spp_transient_stake.clone(),
64              accounts.validator.clone(),
65              accounts.sysvar_clock_id.clone(),
66              accounts.sysvar_rent_id.clone(),
67              accounts.sysvar_stake_history_id.clone(),
68              accounts.sysvar_stake_config_id.clone(),
69              accounts.sysvar_stake_id.clone(),
70          ],
71      )?;
72
73      crate::trace!("Lamports staked");
```



Risk Level:

**Likelihood - 1**

**Impact - 1**

Recommendation:

It is recommended to use correct representative trace messages.

# AUTOMATED TESTING

## 4.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

| ID | package | Short Description |
|---|---|---|
| RUSTSEC-2020-0159 | chrono | Potential segfault in ‘localtime_r‘ invocations |
| RUSTSEC-2020-0071 | time | Potential segfault in the time crate |

THANK YOU FOR CHOOSING

// HALBORN