Access Labs - Access Protocol

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 19th, 2022 - June 29th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/19/2022	Isabel Burruezo
0.2	Draft Review	06/24/2022	Gabi Urrutia
1.0	Remediation Plan	07/01/2022	Isabel Burruezo
1.1	Remediation Plan Review	07/04/2022	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

Access Labs engaged Halborn to conduct a security audit on their programs beginning on May 19th and ending on June 29th. The security assessment was scoped to the programs provided in the access-protocol GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

- Ensure that program functions operate as intended
- Identify potential security issues with the programs

In summary, Halborn identified some security risks that were mostly addressed by the Access Labs 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 the architecture, purpose, and use of the platform.
- Solana program manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger)
- Scanning dependencies for known vulnerabilities (cargo audit).
- Local cluster deployment (solana-test-validator)
- Scanning for common Solana vulnerabilities (soteria)

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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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: access-protocol
   (b) Commit ID: d07ece55ac23bc391a5705d7b4d0d8846a8b095b

Out-of-scope: External libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	3	2	4	3

LIKELIHOOD

IMPACT

		(HAL-01)		
	(HAL-04)			
	(HAL-06) (HAL-07)	(HAL-05)		(HAL-02) (HAL-03)
(HAL-10)	(HAL-08) (HAL-09)			
(HAL-11) (HAL-12)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MULTIPLE VULNERABILITIES IN CHANGE POOL MULTIPLIER INSTRUCTION HANDLING	High	SOLVED - 07/08/2022
(HAL-02) ACCRUING REWARDS WITHOUT BEFORE TOKENS ARE STAKED	High	SOLVED - 06/10/2022
(HAL-03) MISCALCULATION OF REWARDS	High	SOLVED - 06/23/2022
(HAL-04) USER FUNDS LOCKED INDEFINITELY	Medium	SOLVED - 06/04/2022
(HAL-05) IMPOSSIBLE TO CLOSE INACTIVE POOLS	Medium	SOLVED - 06/29/2022
(HAL-06) HARDCODED AUTHORIZED BOND SELLERS ADDRESSES	Low	RISK ACCEPTED
(HAL-07) TRANSFER OWNERSHIP FUNCTIONALITY MISSING	Low	SOLVED - 07/28/2022
(HAL-08) CHECKED ARITHMETIC MISSING	Low	PARTIALLY SOLVED - 06/27/2022
(HAL-09) ZERO AMOUNT CHECK MISSING	Informational	SOLVED - 06/27/2022
(HAL-10) VAULT MINT ADDRESS NOT SYNCED WITH CENTRAL STATE	Informational	RISK ACCEPTED
(HAL-11) SINGLE AUTHORIZED BOND SELLER	Informational	FUTURE RELEASE
(HAL-12) POSSIBLE MISUSE OF HELPER METHODS	Informational	SOLVED - 06/29/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) MULTIPLE VULNERABILITIES IN THE CHANGE POOL MULTIPLIER INSTRUCTION HANDLER - HIGH

Description:

The staker multiplier represents the share of staking rewards that go to stakers, while the owner multiplier represents the remaining share. Both multipliers are used in the calculation of staker and pool rewards.

The ChangePoolMultiplier instruction allows the stake pool owner to change the pool staker's multiplier at any time. However, the instruction handler checks only if the new multiplier value is not greater than 100, and a malicious content publisher could create a pool, wait for stakers to stake their funds and change the multiplier to zero, which would give as a result the following situation:

- the stake pool owner will be able to claim the generated pool rewards based on the owner's multiplier, which is 100% in this case, as the stake multiplier has been changed to zero.
- users will not be able to claim rewards or get their investments back, as they can only unstake after claiming their rewards.

Code Location:

Listing 1: src/processor/change_pool_multiplier.rs (Lines 76,87)

```
66 pub fn process_change_pool_multiplier(
67     program_id: &Pubkey,
68     accounts: &[AccountInfo],
69     params: Params,
70 ) -> ProgramResult {
71     let accounts = Accounts::parse(accounts, program_id)?;
72     let Params { new_multiplier } = params;
```

```
73
74      let mut stake_pool = StakePool::get_checked(accounts.
↳ stake_pool, Tag::StakePool)?;
75
76      if new_multiplier > 100 {
77          msg!("The pool multiplier is a percentage and needs to be
↳ smaller than 100.");
78          return Err(AccessError::Overflow.into());
79      }
80
81      check_account_key(
82          accounts.stake_pool_owner,
83          &Pubkey::new(&stake_pool.header.owner),
84          AccessError::StakeAccountOwnerMismatch,
85      )?;
86
87      stake_pool.header.stakers_part = new_multiplier;
88
89      Ok(())
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to add a check to verify the new multiplier is not less than a proper and fair percentage.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 6f7ab6e918ece665bdba3c7389cee140c5527ed1: A boolean parameter, allow_zero_rewards, has been added to be provided in the call to the ClaimRewards instruction to allow users to decide, in case the pool multiplier is zero, if they want to claim zero rewards and be able to call Unstake and ExecuteUnstake instructions later.

3.2 (HAL-02) ACCRUING REWARDS WITHOUT BEFORE TOKENS ARE STAKED - HIGH

Description:

The StakeAccount is created and the value of its last_claimed_time field is set to the current account creation time.

The Stake instruction allows stakers to stake an amount of tokens.

The ClaimRewards instruction allows stakers to claim their corresponding rewards, the value of last_claimed_time is used to calculate these rewards for the stake account.

However, the last_claimed_time field of the stake account is not updated by the Stake instruction, so a malicious user can create a stake account after the stake pool creation and wait a period of time. During that period, the Crank instruction will have been called many times so that the user can stake an amount of tokens and claim rewards before the next crank. Therefore, that user would get more rewards than they are eligible for, since they will be calculated based on the time period since their staking account was created, which does not match the time they have that staked amount.

Code Location:

Listing 2: src/processor/claim_rewards.rs

```
133  let reward = calc_reward_fp32(
134          current_time,
135          stake_account.last_claimed_time,
136          &stake_pool,
137          true,
138      )?
139      // Multiply by the staker shares of the total pool
140      .checked_mul(stake_account.stake_amount as u128)
```

```
141     .map(|r| r >> 32)
142     .and_then(safe_downcast)
143     .ok_or(AccessError::Overflow)?;
```

Listing 3: src/processor/stake.rs

```
151     assert_valid_fee(accounts.fee_account, &central_state.
 ↳ authority)?;
152     let fees = (amount * FEES) / 100;
153     amount -= fees;
154
155     if stake_account.stake_amount > 0
156         && stake_account.last_claimed_time < stake_pool.header.
 ↳ last_crank_time
157     {
158         return Err(AccessError::UnclaimedRewards.into());
159     }
160     // Transfer tokens
161     let transfer_instruction = transfer(
162         &spl_token::ID,
163         accounts.source_token.key,
164         accounts.vault.key,
165         accounts.owner.key,
166         &[],
167         amount,
168     )?;
169     invoke(
170         &transfer_instruction,
171         &[
172             accounts.spl_token_program.clone(),
173             accounts.source_token.clone(),
174             accounts.vault.clone(),
175             accounts.owner.clone(),
176         ],
177     )?;
178
179     // Transfer fees
180     let transfer_fees = transfer(
181         &spl_token::ID,
182         accounts.source_token.key,
183         accounts.fee_account.key,
184         accounts.owner.key,
185         &[],
186         fees,
```

```
187     )?;
188     invoke(
189         &transfer_fees,
190         &[
191             accounts.spl_token_program.clone(),
192             accounts.source_token.clone(),
193             accounts.fee_account.clone(),
194             accounts.owner.clone(),
195         ],
196     )?;
197 
198     if stake_account
199         .stake_amount
200         .checked_add(amount)
201         .ok_or(AccessError::Overflow)?
202         < std::cmp::min(
203             stake_account.pool_minimum_at_creation,
204             stake_pool.header.minimum_stake_amount,
205         )
206     {
207         msg!(
208             "The minimum stake amount must be > {}",
209             stake_account.pool_minimum_at_creation
210         );
211         return Err(ProgramError::InvalidArgument);
212     }
213 
214     // Update stake account
215     stake_account.deposit(amount)?;
216     stake_pool.header.deposit(amount)?;
217 
218     //Update central state
219     central_state.total_staked = central_state
220         .total_staked
221         .checked_add(amount)
222         .ok_or(AccessError::Overflow)?;
223 
224     // Save states
225     stake_account.save(&mut accounts.stake_account.data.borrow_mut
 ↳ ());
226     central_state.save(&mut accounts.central_state_account.data.
 ↳ borrow_mut());
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

It is recommended to update the last_claimed_time of the stake account when its owner is staking and its stake_amount was zero before.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 307fa9fd5618d3063aea012eee2881972dcc006f: Added a check in staker instruction handler to verify if the account's stake amount so far is zero and if it is, its last_claimed_time value is updated.

3.3 (HAL-03) MISCALCULATION OF REWARDS - HIGH

Description:

Stakers can claim their rewards by sending the ClaimRewards instruction daily after the crank, or claim the total amount in one go later.

However, stakers who claim rewards daily get a higher total amount since the reward calculation loop always performed an extra iteration in that case. For example, if a staker claims daily for 7 days, the amount of rewards they will have earned on day 7 will be greater than the amount collected by the staker who claims them all at once on day 7.

Code Location:

Listing 4: src/processor/claim_rewards.rs (Line 133)

```
133 let reward = calc_reward_fp32(
134         current_time,
135         stake_account.last_claimed_time,
136         &stake_pool,
137         true,
138     )?
139     // Multiply by the staker shares of the total pool
140     .checked_mul(stake_account.stake_amount as u128)
141     .map(|r| r >> 32)
142     .and_then(safe_downcast)
143     .ok_or(AccessError::Overflow)?;
```

Listing 5: src/utils.rs (Lines 35,40)

```
14 pub fn calc_reward_fp32(
15     current_time: i64,
16     last_claimed_time: i64,
17     stake_pool: &StakePoolRef,
18     staker: bool,
19 ) -> Result<u128, ProgramError> {
20     let mut nb_days_to_claim =
```

```
21          current_time.saturating_sub(last_claimed_time) as u64 /
↳ SECONDS_IN_DAY;
22      msg!("Nb of days behind {}", nb_days_to_claim);
23      nb_days_to_claim = std::cmp::min(nb_days_to_claim,
↳ STAKE_BUFFER_LEN - 1);
24
25      if current_time
26          .checked_sub(stake_pool.header.last_crank_time)
27          .ok_or(AccessError::Overflow)?
28          > SECONDS_IN_DAY as i64
29      {
30          #[cfg(not(feature = "no-lock-time"))]
31          return Err(AccessError::PoolMustBeCranked.into());
32      }
33
34      // Saturating as we don't want to wrap around when there haven
↳ 't been sufficient cranks
35      let mut i = (stake_pool.header.current_day_idx as u64).
↳ saturating_sub(nb_days_to_claim)
36          % STAKE_BUFFER_LEN;
37
38      // Compute reward for all past days
39      let mut reward: u128 = 0;
40       while i != (stake_pool.header.current_day_idx as u64 + 1) %
↳ STAKE_BUFFER_LEN {
41          let curr_day_reward = if staker {
42              stake_pool.balances[i as usize].stakers_reward
43          } else {
44              stake_pool.balances[i as usize].pool_reward
45          };
46          reward = reward
47              .checked_add(curr_day_reward)
48              .ok_or(AccessError::Overflow)?;
49          i = (i + 1) % STAKE_BUFFER_LEN;
50      }
51
52      if reward == 0 {
53          msg!("No rewards to claim, no operation.");
54          return Err(AccessError::NoOp.into());
55      }
56
57      Ok(reward)
58
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

It is recommended to update the initial index in the reward calculation loop to eliminate the extra iteration.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 149ad1baf1d2033700d07f3c22515bc9df0ed27b: The initial index in the rewards' calculation loop has been updated to start in the proper one.

3.4 (HAL-04) USER FUNDS LOCKED INDEFINITELY - MEDIUM

Description:

The CloseStakePool instruction allows the stake pool owner to close the pool account at any time, as long as the pool is active and the value of stake_pool.header.total_stake is zero.

However, the stake pool vault balance is not checked to be zero, so the stake pool can be closed even if there are users who initialized the unstake operation with the Unstake instruction but have not executed the unstake with the ExecuteUnstake instruction yet, in which case those users will not be able to recover their investment.

Code Location:

Listing 6: src/processor/close_stake_pool.rs (Line 79)

```
64 pub fn process_close_stake_pool(
65     program_id: &Pubkey,
66     accounts: &[AccountInfo],
67     _params: Params,
68 ) -> ProgramResult {
69     let accounts = Accounts::parse(accounts, program_id)?;
70
71     let mut stake_pool = StakePool::get_checked(accounts.
↳ stake_pool_account, Tag::StakePool)?;
72
73     check_account_key(
74         accounts.owner,
75         &Pubkey::new(&stake_pool.header.owner),
76         AccessError::WrongStakePoolOwner,
77     )?;
78
79     assert_empty_stake_pool(&stake_pool)?;
80
81     stake_pool.header.close();
82
```

```
83      let mut stake_pool_lamports = accounts.stake_pool_account.
↳ lamports.borrow_mut();
84      let mut owner_lamports = accounts.owner.lamports.borrow_mut();
85
86      **owner_lamports += **stake_pool_lamports;
87      **stake_pool_lamports = 0;
```

Listing 7: src/utils.rs (Line 88)

```
87 pub fn assert_empty_stake_pool(stake_pool: &StakePoolRef) ->
↳ ProgramResult {
88      if stake_pool.header.total_staked != 0 {
89          msg!("The stake pool must be empty");
90          return Err(AccessError::StakePoolMustBeEmpty.into());
91      }
92      Ok(())
93 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to add a check in the process_close_stake_pool function to verify the vault balance is zero before closing the pool account.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 8f71aee18b57e16f9d24e8fc163f6fe822cd607c: Added a check in the close_stake_pool instruction handler to verify that the vault is empty, and if it is not, it does not allow its owner to close it.

3.5 (HAL-05) IMPOSSIBLE TO CLOSE INACTIVE POOLS - MEDIUM

Description:

The CloseStakePool instruction allows the stake pool owner to close an active stake pool if the total amount staked is zero. If a stake pool is misconfigured, it cannot be closed until activated by the central state authority.

However, a user could stake in an activated misconfigured pool before the owner manages to close it, which means that the owner would need to wait for the whole process of staking, claiming and unstaking to finish.

Code Location:

Listing 8: src/processor/close_stake_pool.rs (Line 71)

```
64 pub fn process_close_stake_pool(
65     program_id: &Pubkey,
66     accounts: &[AccountInfo],
67     _params: Params,
68 ) -> ProgramResult {
69     let accounts = Accounts::parse(accounts, program_id)?;
70 
71     let mut stake_pool = StakePool::get_checked(accounts.
↳ stake_pool_account, Tag::StakePool)?;
72 
73     check_account_key(
74         accounts.owner,
75         &Pubkey::new(&stake_pool.header.owner),
76         AccessError::WrongStakePoolOwner,
77     )?;
78 
79 
80     assert_empty_stake_pool(&stake_pool)?;
81 
82     stake_pool.header.close();
83 
84     let mut stake_pool_lamports = accounts.stake_pool_account.
```

```
↳ lamports.borrow_mut();
85
86      let mut owner_lamports = accounts.owner.lamports.borrow_mut();
87
88      **owner_lamports += **stake_pool_lamports;
89      **stake_pool_lamports = 0;
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Implement a feature to allow the pool owner to close misconfigured inactive pools.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 2a03cfd2bc8f1121391ed78b692d7eb2deca3ca8: A vector of tags for stake pool status checking has been added. In the CloseStakePool instruction handler, the check has also been modified to allow the stake pool to be closed if it is also inactive.

3.6 (HAL-06) HARDCODED AUTHORIZED BOND SELLERS ADDRESSES - LOW

Description:

The create_bond instruction handler checks the signer's account address is in the hardcoded AUTHORIZED_BOND_SELLERS array at the user-supplied index.

The AUTHORISED_BOND_SELLERS array contains account addresses of authorized bond sellers who create and sign bonds.

The addresses in the AUTHORISED_BOND_SELLERS array cannot be changed without redeploying the program if any of those accounts is compromised.

Code Location:

Listing 9: src/processor/create_bond.rs (Line 117)

```
 97  pub fn process_create_bond(
 98      program_id: &Pubkey,
 99      accounts: &[AccountInfo],
100      params: Params,
101  ) -> ProgramResult {
102      let accounts = Accounts::parse(accounts, program_id)?;
103
104      let (derived_key, nonce) =
105          BondAccount::create_key(&params.buyer, params.
 ↳ total_amount_sold, program_id);
106
107      let stake_pool = StakePool::get_checked(accounts.stake_pool,
 ↳ Tag::StakePool)?;
108
109      check_account_key(
110          accounts.bond_account,
111          &derived_key,
112          AccessError::AccountNotDeterministic,
113      )?;
114      assert_uninitialized(accounts.bond_account)?;
115
```

```
116     #[cfg(not(feature = "no-bond-signer"))]
117     assert_authorized_seller(accounts.seller, params.seller_index
 ↳ as usize)?;
```

Listing 10: src/utils.rs (Line 137)

```
136 pub fn assert_authorized_seller(seller: &AccountInfo, seller_index
 ↳ : usize) -> ProgramResult {
137     let expected_seller = AUTHORIZED_BOND_SELLERS
138         .get(seller_index)
139         .ok_or(AccessError::UnauthorizedSeller)?;
140     if seller.key != expected_seller {
141         return Err(AccessError::UnauthorizedSeller.into());
142     }
143     Ok(())
```

Listing 11: src/state.rs

```
503 /// List of authorized bond sellers
504 pub const AUTHORIZED_BOND_SELLERS: [Pubkey; 1] = [solana_program::
 ↳ pubkey!(
505     "3Nrq6mCNL5i8Qk4APhggbwXismcsF23gNVDEaKycZBL8"
506 )];
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Implement a governance function to update the contents of the AUTHORIZED_BOND_SELLERS array.

Remediation Plan:

RISK ACCEPTED: The Access Labs team accepted the risk of this finding.

3.7 (HAL-07) TRANSFER OWNERSHIP FUNCTIONALITY MISSING - LOW

Description:

The program lacks the option to update the central state authority address. If the authority account is compromised, or if the development team needs to change the address for operational reasons, a significant portion of the contract's functionality will become unusable.

Code Location:

Listing 12: src/processor/create_central_state.rs

```
73 pub fn process_create_central_state(
74     program_id: &Pubkey,
75     accounts: &[AccountInfo],
76     params: Params,
77 ) -> ProgramResult {
78     let accounts = Accounts::parse(accounts)?;
79     let (derived_state_key, nonce) = CentralState::find_key(
↳ program_id);
80
81     check_account_key(
82         accounts.state_account,
83         &derived_state_key,
84         AccessError::AccountNotDeterministic,
85     )?;
86
87     let state = CentralState::new(
88         nonce,
89         params.daily_inflation,
90         *accounts.mint.key,
91         params.authority,
92         0,
93     );
94
95     Cpi::create_account(
96         program_id,
97         accounts.system_program,
```

```
98          accounts.fee_payer,
99          accounts.state_account,
100         &[&program_id.to_bytes(), &[nonce]],
101         state.borsh_len(),
102     )?;
103
104     state.save(&mut accounts.state_account.data.borrow_mut());
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to add **authority transfer** capabilities to the program, split into two different functions: `set_authority` and `accept_authority`. The latter function allows the transfer to be completed by the recipient, which protects the program against possible typing errors compared to one-step **authority** change features.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit `db411744cb8c2514b7837c90db0af8bf5b05ba67`: The `change_central_state_authority` instruction has been added to include the capability to the program to transfer the ownership of the central state.

3.8 (HAL-08) CHECKED ARITHMETIC MISSING - LOW

Description:

Unsafe arithmetic operations were identified in multiple files and program functions.

Code Location:

Listing 13: src/state.rs (Line 651)

```
649  pub fn calc_unlock_amount(&self, missed_periods: u64) -> Result<
 ↳ u64, ProgramError> {
650          msg!("Missed periods {}", missed_periods);
651          let cumulated_unlock_amnt = missed_periods * self.
 ↳ unlock_amount;
652          msg!(
653              "Unlock amount {} Total amount {}",
654              cumulated_unlock_amnt,
655              self.total_amount_sold
656          );
```

Listing 14: src/processor/close_stake_account.rs (Lines 76,77)

```
73      let mut stake_lamports = accounts.stake_account.lamports.
 ↳ borrow_mut();
74      let mut owner_lamports = accounts.owner.lamports.borrow_mut();
75
76      **owner_lamports += **stake_lamports;
77      **stake_lamports = 0;
```

Listing 15: src/processor/close_stake_pool.rs (Lines 85,86)

```
80      stake_pool.header.close();
81
82      let mut stake_pool_lamports = accounts.stake_pool_account.
 ↳ lamports.borrow_mut();
83      let mut owner_lamports = accounts.owner.lamports.borrow_mut();
```

```
84
85      **owner_lamports += **stake_pool_lamports;
86      **stake_pool_lamports = 0;
```

Listing 16: src/processor/stake.rs (Lines 153,154)

```
151     assert_valid_fee(accounts.fee_account, &central_state.
↳ authority)?;
152
153     let fees = (amount * FEES) / 100;
154     amount -= fees;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider using checked arithmetic operations instead of regular arithmetic operators to handle this gracefully.

Remediation Plan:

PARTIALLY SOLVED: The Access Labs team fixed partially this issue in commit 6f0cbf6ee9de741c67ff053ae41176febad99265: Regular arithmetic operators in state has been changed for using checked arithmetic operations.

3.9 (HAL-09) ZERO AMOUNT CHECK MISSING - LOW

Description:

The Stake instruction handler allows users to stake ACCESS in pools. The instruction handler checks whether the total amount staked by this account so far account is above the threshold.

However, because when the threshold is reached the function does not validate the user-supplied amount to be greater than 0, users can execute this instruction successfully without actually staking ACCESS.

Code Location:

Listing 17: /src/processor/stake.rs

```
205     if stake_account
206         .stake_amount
207         .checked_add(amount)
208         .ok_or(AccessError::Overflow)?
209         < std::cmp::min(
210             stake_account.pool_minimum_at_creation,
211             stake_pool.header.minimum_stake_amount,
212         )
213     {
214         msg!(
215             "The minimum stake amount must be > {}",
216             stake_account.pool_minimum_at_creation
217         );
218         return Err(ProgramError::InvalidArgument);
219     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add a check to verify the amount provided to stake is greater than zero.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 6f0cbf6ee9de741c67ff053ae41176febad99265: A check has been added in stake instruction handler to prevent staking of an amount equal to zero.

3.10 (HAL-10) VAULT MINT ADDRESS NOT SYNCED WITH CENTRAL STATE - INFORMATIONAL

Description:

The CreateStakePool instruction allows the content publisher to create a betting pool by providing a number of accounts, including the token vault. Its parameters are verified by the instruction handler. However, its mint is checked to match the encoded ACCESS_MINT address and not the central state mint, which is the actual token mint used in transfers.

Code Location:

Listing 18: /src/processor/create_stake_pool.rs (Line 94)

```
79 pub fn process_create_stake_pool(
80     program_id: &Pubkey,
81     accounts: &[AccountInfo],
82     params: Params,
83 ) -> ProgramResult {
84     let accounts = Accounts::parse(accounts)?;
85 
86     let (derived_stake_key, nonce) = StakePool::find_key(&params.
↳ owner, program_id);
87 
88     check_account_key(
89         accounts.stake_pool_account,
90         &derived_stake_key,
91         AccessError::AccountNotDeterministic,
92     )?;
93 
94     assert_valid_vault(accounts.vault, &derived_stake_key)?;
```

Listing 19: /src/utils.rs (Line 113)

```
103 pub fn assert_valid_vault(account: &AccountInfo, vault_signer: &
↳ Pubkey) -> ProgramResult {
```

```
104     let acc = Account::unpack(&account.data.borrow())?;
105     if &acc.owner != vault_signer {
106         msg!("The vault account should be owned by the stake pool
 ↳ signer");
107         return Err(ProgramError::InvalidArgument);
108     }
109     if acc.close_authority.is_some() || acc.delegate.is_some() {
110         msg!("Invalid vault account provided");
111         return Err(ProgramError::InvalidArgument);
112     }
113     if acc.mint != ACCESS_MINT {
114         msg!("Invalid ACCESS mint");
115         #[cfg(not(feature = "no-mint-check"))]
116         return Err(ProgramError::InvalidArgument);
117     }
118     Ok(())
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to check if the mint of the vault matches the mint of the central state.

Remediation Plan:

RISK ACCEPTED: The Access Labs team accepted the risk of this finding.

3.11 (HAL-11) SINGLE AUTHORIZED BOND SELLER - INFORMATIONAL

Description:

A bond can be created by a single authorized seller, but in order to actually sell it, all authorized sellers must sign the SignBond instruction.
However, in the current implementation only one authorized seller exists, so when a bond is created, a single signature is sufficient to authorize its sale, centralising the operation.

Code Location:

Listing 20: src/state.rs

```
500 /// Number of sellers who need to agree for a bond to be sold
501 pub const BOND_SIGNER_THRESHOLD: u64 = 1;
502
503 /// List of authorized bond sellers
504 pub const AUTHORIZED_BOND_SELLERS: [Pubkey; 1] = [solana_program::
 ↳ pubkey!(
505     "3Nrq6mCNL5i8Qk4APhggbwXismcsF23gNVDEaKycZBL8"
506 )];
```

Listing 21: src/processor/sign_bond.rs

```
54 pub fn process_sign_bond(
55     program_id: &Pubkey,
56     accounts: &[AccountInfo],
57     params: Params,
58 ) -> ProgramResult {
59     let accounts = Accounts::parse(accounts, program_id)?;
60     let mut bond = BondAccount::from_account_info(accounts.
 ↳ bond_account, true)?;
61     assert_authorized_seller(accounts.seller, params.seller_index
 ↳ as usize)?;
62
63     if bond.sellers.len() == BOND_SIGNER_THRESHOLD as usize {
```

```
64          msg!("There are enough signers already");
65          return Err(AccessError::NoOp.into());
66      }
67
68      #[cfg(not(feature = "no-bond-signer"))]
69      for current_seller in &bond.sellers {
70          if accounts.seller.key == current_seller {
71              msg!("The seller has already signed");
72              return Err(AccessError::BondSellerAlreadySigner.into()
↳ );
73          }
74      }
75
76      bond.sellers.push(*accounts.seller.key);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to authorize multiple sellers so that sales have to be approved by multiple parties.

Remediation Plan:

PENDING: The Access Labs team will fix this issue be following the recommendation in a future version of the code when the program is deployed.

3.12 (HAL-12) POSSIBLE MISUSE OF HELPER METHODS - INFORMATIONAL

Description:

The intention and use of helper methods in Rust, like `unwrap`, is very useful for testing environments because a value is forcibly demanded to get an error (aka `panic!`) if the `Option` the methods is called on doesn't have `Some` value or `Result`. Nevertheless, leaving `unwrap` functions in production environments is a bad practice because not only will this cause the program to crash out, or panic!. In addition, no helpful messages are shown to help the user solve, or understand the reason of the error.

Code Location:

Note: some usages of `unwrap` are justified and were excluded from the listing below.

Listing 22

```
src/processor/claim_rewards.rs:111:    let current_time = Clock::
↳ get().unwrap().unix_timestamp;
src/processor/claim_pool_rewards.rs:101:    let current_time =
↳ Clock::get().unwrap().unix_timestamp;
src/processor/claim_bond_rewards.rs:113:    let current_time =
↳ Clock::get().unwrap().unix_timestamp;
src/state.rs:162:                try_cast_slice_mut(rem).unwrap(),
src/state.rs:219:            .checked_add(nb_days_passed.try_into
↳ ().unwrap())
src/state.rs:231:        Pubkey::create_program_address(seeds,
↳ program_id).unwrap()
src/state.rs:253:            last_crank_time: Clock::get().unwrap
↳ ().unix_timestamp,
src/state.rs:254:            last_claimed_time: Clock::get().
↳ unwrap().unix_timestamp,
src/state.rs:269:        self.total_staked = self.total_staked.
↳ checked_add(amount).unwrap();
```

```
10 src/state.rs:274:        self.total_staked = self.total_staked.
↳ checked_sub(amount).unwrap();
11 src/state.rs:359:        Pubkey::create_program_address(seeds,
↳ program_id).unwrap()
12 src/state.rs:372:        self.serialize(&mut dst).unwrap()
13 src/state.rs:389:        self.stake_amount = self.stake_amount.
↳ checked_add(amount).unwrap();
14 src/state.rs:395:        self.stake_amount = self.stake_amount.
↳ checked_sub(amount).unwrap();
15 src/state.rs:478:        Pubkey::create_program_address(
↳ signer_seeds, program_id).unwrap()
16 src/state.rs:486:        self.serialize(&mut dst).unwrap()
17 src/state.rs:619:        self.serialize(&mut dst).unwrap()
18
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not use the unwrap function in production environment because this use provokes panic! and may crash the contract without verbose error messages. Crashing the system will result in a loss of availability, and in some cases, even private information stored in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrap or using the error-chain crate for errors.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 8b0e0d49c3bbf72f3e101427116873d40671eb25: The use of unwrap function has been replaced by propagating the error with ?.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Description:

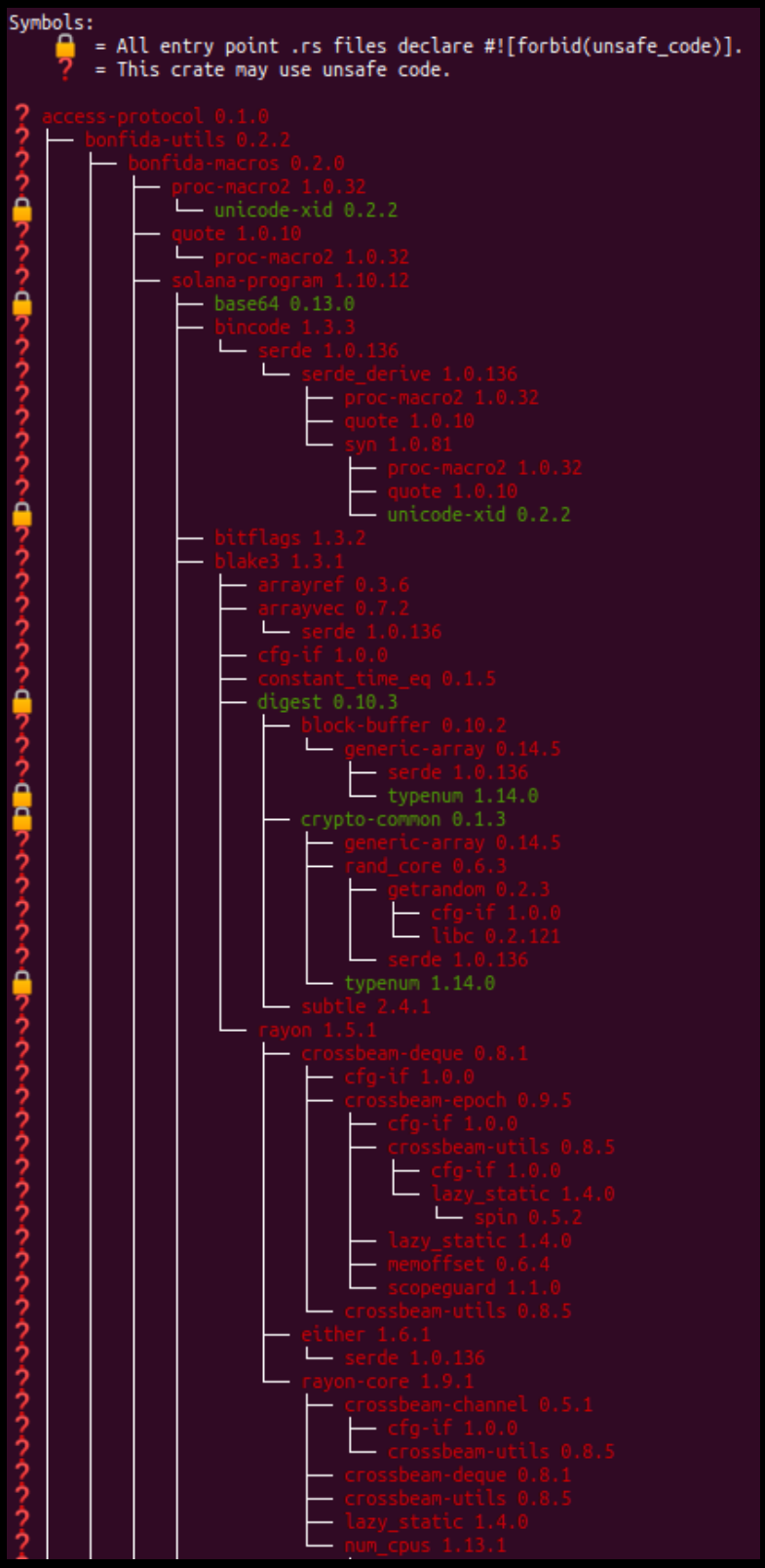
Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo-audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in 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. Only security detections are 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.

Results cargo-audit:

ID	package	Short Description
RUSTSEC-2022-0013	regex	Regexes with large repetitions on empty sub-expressions take a very long time to parse

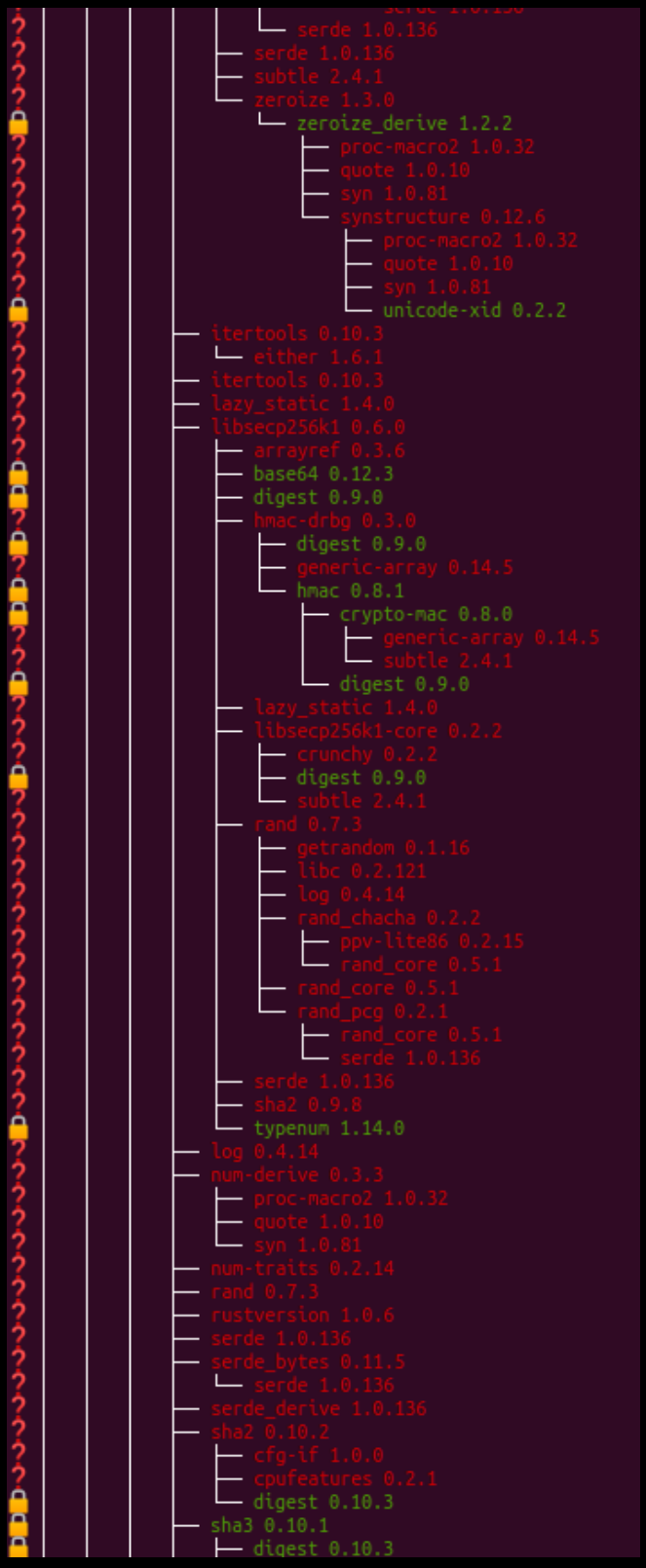
AUTOMATED TESTING

Result cargo-geiger:

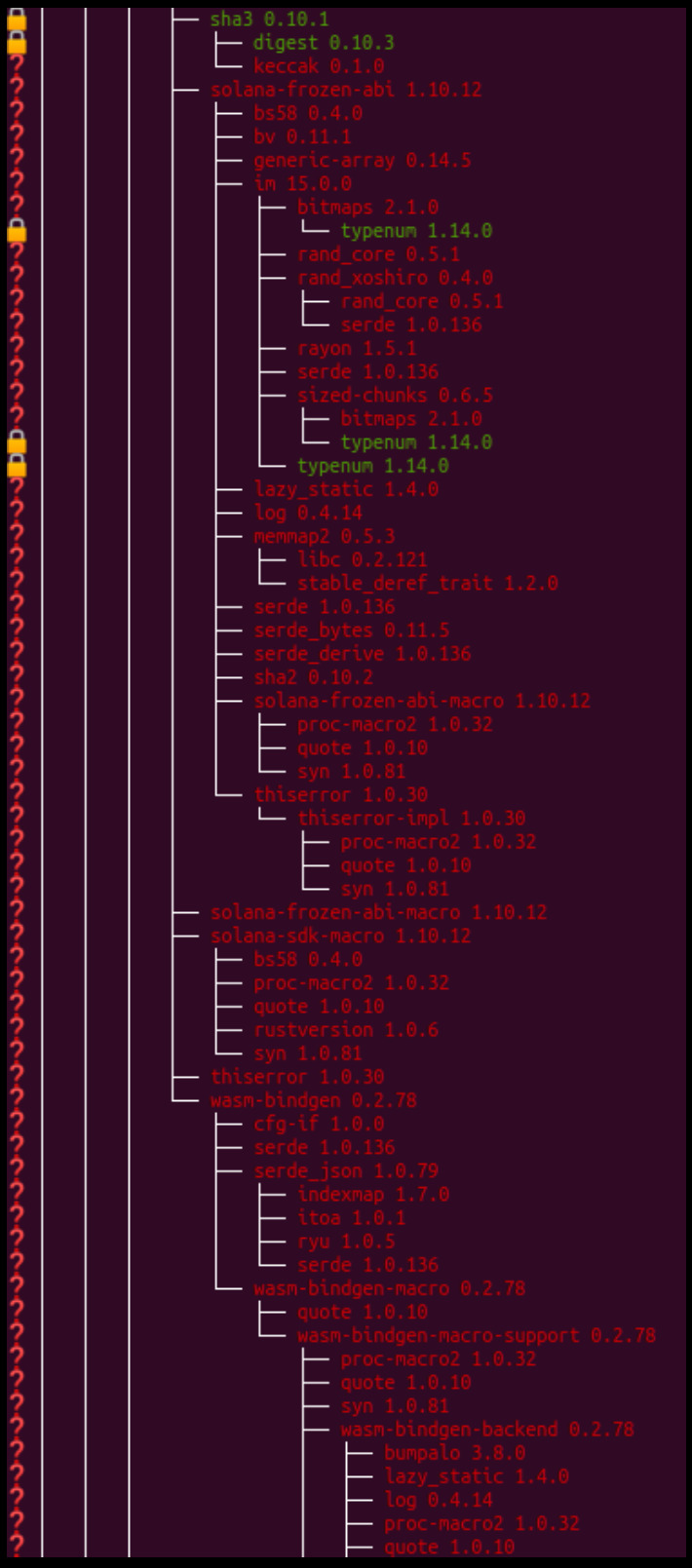


AUTOMATED TESTING

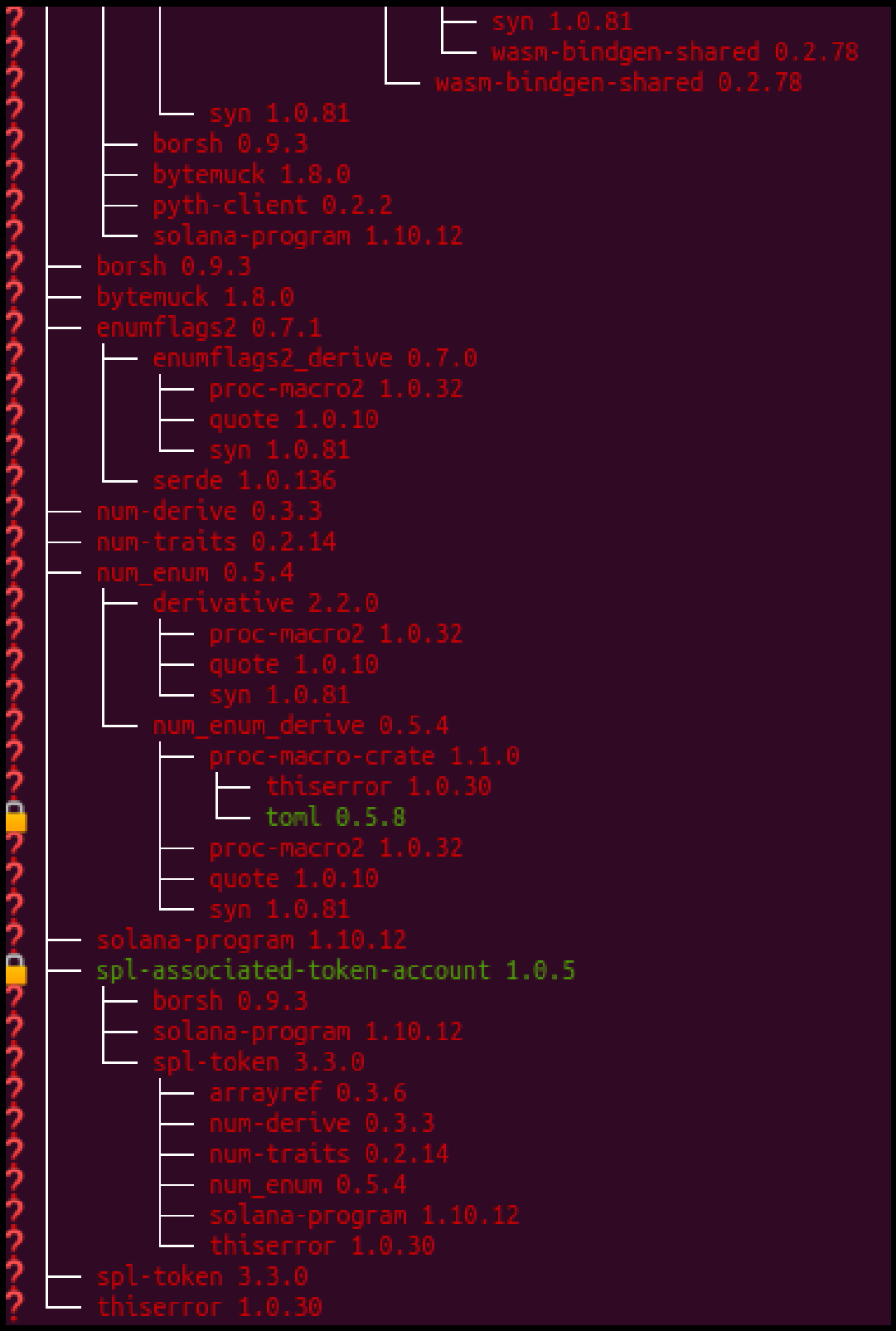
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



4.2 AUTOMATED VULNERABILITY SCANNING

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was Soteria, a security analysis service for Solana programs. Soteria performed a scan on all the programs and sent the compiled results to the analyzers to locate any vulnerabilities.

Results:

Soteria scanner found four unsafe arithmetic operations that were reported with HAL-06 vulnerability in previous chapter.

AUTOMATED TESTING

```
Cargo.toml: spl_token version: 3.3.0
anchor_lang_version: 3.3.0 anchorVersionTooOld: 0
 - ✔ [00m:01s] Loading IR From File
 - ▖ [00m:00s] Running Compiler Optimization Passes
EntryPoints:
entrypoint
 - ✔ [00m:00s] Running Compiler Optimization Passes
 - ✔ [00m:00s] Running Pointer Analysis
=============This arithmetic operation may be UNSAFE!===============
Found a potential vulnerability at line 651, column 37 in src/state.rs
The mul operation may result in overflows:

 645|        let result = BondAccount::deserialize(&mut data)?;
 646|        Ok(result)
 647|    }
 648|
 649|    pub fn calc_unlock_amount(&self, missed_periods: u64) -> Result<u64, ProgramError> {
 650|        msg!("Missed periods {}", missed_periods);
>651|        let cumulated_unlock_amnt = missed_periods * self.unlock_amount;
 652|        msg!(
 653|            "Unlock amount {} Total amount {}",
 654|            cumulated_unlock_amnt,
 655|            self.total_amount_sold
 656|        );
 657|
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h09a2235c077ce828 [src/entrypoint.rs:21]
>>>  access_protocol::processor::unlock_bond_tokens::process_unlock_bond_tokens::he388d04d36e054b3 [src/processor.rs:136]
>>>    access_protocol::state::BondAccount::calc_unlock_amount::h387093bc88aea638 [src/processor/unlock_bond_tokens.rs:151]

=============This arithmetic operation may be UNSAFE!===============
Found a potential vulnerability at line 76, column 5 in src/processor/close_stake_account.rs
The add operation may result in overflows:

 70|    stake_account.close();
 71|    stake_account.save(&mut accounts.stake_account.data.borrow_mut());
 72|
 73|    let mut stake_lamports = accounts.stake_account.lamports.borrow_mut();
 74|    let mut owner_lamports = accounts.owner.lamports.borrow_mut();
 75|
>76|    **owner_lamports += **stake_lamports;
 77|    **stake_lamports = 0;
 78|
 79|    Ok(())
 80|}
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h09a2235c077ce828 [src/entrypoint.rs:21]
>>>  access_protocol::processor::close_stake_account::process_close_stake_account::hbd2c0c6f40eba130 [src/processor.rs:112]
```

AUTOMATED TESTING

```
=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 85, column 5 in src/processor/close_stake_pool.rs
The add operation may result in overflows:

 79|
 80|    stake_pool.header.close();
 81|
 82|    let mut stake_pool_lamports = accounts.stake_pool_account.lamports.borrow_mut();
 83|    let mut owner_lamports = accounts.owner.lamports.borrow_mut();
 84|
>85|    **owner_lamports += **stake_pool_lamports;
 86|    **stake_pool_lamports = 0;
 87|
 88|    Ok(())
 89|}
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h09a2235c077ce828 [src/entrypoint.rs:21]
>>>  access_protocol::processor::close_stake_pool::process_close_stake_pool::h903e5491a5c79de4 [src/processor.rs:108]

=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 154, column 5 in src/processor/stake.rs
The sub operation may result in underflows:

 148|        AccessError::StakePoolVaultMismatch,
 149|    )?;
 150|
 151|    assert_valid_fee(accounts.fee_account, &central_state.authority)?;
 152|
 153|    let fees = (amount * FEES) / 100;
>154|    amount -= fees;
 155|
 156|    if stake_account.stake_amount > 0
 157|        && stake_account.last_claimed_time < stake_pool.header.last_crank_time
 158|    {
 159|        return Err(AccessError::UnclaimedRewards.into());
 160|    }
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h09a2235c077ce828 [src/entrypoint.rs:21]
>>>  access_protocol::processor::stake::process_stake::h7cd53f6f71781ef2 [src/processor.rs:73]

 - ✔[00m:00s] Building Static Happens-Before Graph
 - ✔[00m:00s] Detecting Vulnerabilities
detected 0 untrustful accounts in total.
detected 4 unsafe math operations in total.

--------The summary of potential vulnerabilities in all.ll--------

        4 unsafe arithmetic issues
```

THANK YOU FOR CHOOSING

// HALBORN