HALBORN

Access Labs - Access Protocol Updates

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 16th, 2023 - January 19th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/16/2023	Isabel Burruezo
0.2	Draft Updates	01/19/2023	Isabel Burruezo
0.3	Draft Review	01/19/2023	Piotr Cielas
0.4	Draft Review	01/20/2023	Gabi Urrutia
1.0	Remediation Plan	01/25/2023	Isabel Burruezo
1.1	Remediation Plan Review	01/25/2023	Piotr Cielas
1.2	Remediation Plan Review	01/25/2023	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Isabel Burruezo	Halborn	Isabel.Burruezo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Access Protocol offers a new model monetization layer for all digital content creators. It is a Web3 protocol built on Solana and Starknet that offers an alternative to B2C subscriptions.

Access Labs engaged Halborn to conduct a security audit on their program, beginning on January 16th, 2023 and ending on January 19th, 2023 .

The security audit 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 a week for the engagement and assigned a full-time security engineer to audit the security of the program in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and Solana program hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Identify potential security issues within the program

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were addressed by Access Labs .

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the Solana 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.
- Program manual code review and walkthrough to identify logic issues.
- Mapping out possible attack vectors
- 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 runtime testing (solana-test-framework)
- 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.

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

1.4 SCOPE

1. Access Protocol

- Repository: access-protocol
- Diff in scope:
    1. b73a5b231c6672d79e2fe5b7493ca4e675219d9..7b8a9d6129c4c0e43e0e0d6bad97e6074
- Programs in scope:
    1. access-protocol (access-protocol/smart-contract/program)

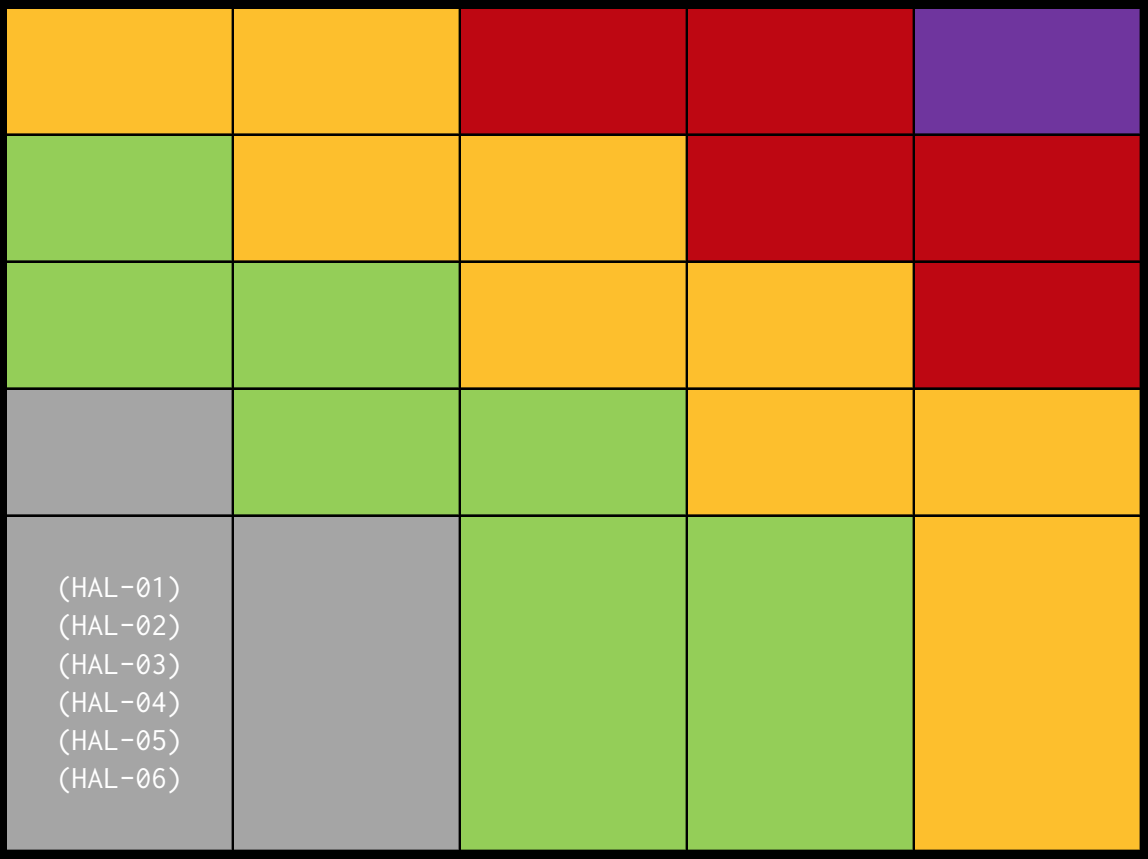
Out-of-scope: External libraries, dependencies and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	6

LIKELIHOOD

IMPACT



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SUSCEPTIBLE TO INTEGER OVERFLOW	Informational	SOLVED - 24/01/2023
(HAL-02) OWNER OF SOURCE TOKEN CHECK MISSING DURING STAKING	Informational	SOLVED - 01/24/2023
(HAL-03) MINT OF DESTINATION ACCOUNT CHECK MISSING	Informational	SOLVED - 01/24/2023
(HAL-04) MISLEADING ACCESS ERROR	Informational	SOLVED - 01/25/2023
(HAL-05) MISSING CARGO OVERFLOW CHECKS	Informational	SOLVED - 01/24/2023
(HAL-06) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	PARTIALLY SOLVED - 01/25/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) SUSCEPTIBLE TO INTEGER OVERFLOW - INFORMATIONAL

Description:

Integer overflow/underflow occurs when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented by a given number of bits, either greater than the maximum or less than the minimum representable value. Although integer overflows and underflows do not cause Rust to panic in the release mode, the consequences could be dire if the result of those operations is used in financial calculations.

The Stake instruction handler could be affected by an overflow, causing legitimate transactions to fail and thus cause a denial of service for the users.
The stake_amount is added to the amount_in_bonds and is compared against a value (pool_minimum_at_creation). If the sum of both values overflow, the comparison returns false and the transaction fails.

Code Location:

Listing 1: holding/src/create_bond.rs (Line 32)

```
25 /// The required parameters for the `create_bond` instruction
26 pub struct Params {
27     /// Ultimate buyer of the bond
28     pub buyer: Pubkey,
29     /// Total amount of ACCESS tokens being sold
30     pub total_amount_sold: u64,
31     /// Total price of the bond
32     pub total_quote_amount: u64,
33     /// Mint of the token used to buy the bond
```

Listing 2: holding/src/state.rs (Line 547)

```
24     #[allow(clippy::too_many_arguments)]
25     pub fn new(
26         owner: Pubkey,
```

```
27          total_amount_sold: u64,
28          total_quote_amount: u64,
29          quote_mint: Pubkey,
30          seller_token_account: Pubkey,
31          unlock_start_date: i64,
32          unlock_period: i64,
33          unlock_amount: u64,
34          last_unlock_time: i64,
```

Listing 3: holding/src/stake.rs (Line 27)

```
24 /// The required parameters for the `stake` instruction
25 pub struct Params {
26     // Amount to stake
27     pub amount: u64,
28 }
```

Listing 4: holding/src/stake.rs (Line 178)

```
164  if let Some(bond_account) = accounts.bond_account {
165          let bond_account = BondAccount::from_account_info(
 ↳ bond_account, false)?;
166
167          check_account_key(
168              accounts.owner,
169              &bond_account.owner,
170              AccessError::WrongOwner,
171          )?;
172          check_account_key(
173              accounts.stake_pool,
174              &bond_account.stake_pool,
175              AccessError::StakePoolMismatch,
176          )?;
177
178          amount_in_bonds = bond_account.total_staked;
179      }
180
181      // if we were previously under the minimum stake limit it gets
 ↳  reset to the pool's one
182
183      if stake_account.stake_amount + amount_in_bonds <
 ↳ stake_account.pool_minimum_at_creation {
184          stake_account.pool_minimum_at_creation = stake_pool.header
```

```
  ↳ .minimum_stake_amount;
185     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use safe and verified math libraries such as checked_add for consistent arithmetic operations throughout the Solana program system.
Consider using Rust safe arithmetic functions for primitives instead of standard arithmetic operators. You may also want to either

- Allow users to stake std::u64::MAX as maximum value
- Allow sellers to create bond with std::u64::MAX maximum total_quote_amount value.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 8e0f7854de2dc4216409719f2aa982315467119a: the addition operator was replaced with the checked_add function.

3.2 (HAL-02) OWNER AND MINT OF SOURCE TOKEN CHECK MISSING DURING STAKING - INFORMATIONAL

Description:

The Stake instruction allows stakers to stake an amount of tokens. For this purpose, it is necessary to provide different accounts, among which is the source_token account.
Stakers deposit their tokens to the stake pool vault.

The instruction handler is checking if the source_token account is really a token account. However, it does not check if the owner of this account matches the owner account provided.

Code Location:

Listing 5: src/processor/stake.rs (Line 79)

```
69 pub fn parse(
70     accounts: &'a [AccountInfo<'b>],
71     program_id: &Pubkey,
72 ) -> Result<Self, ProgramError> {
73     let accounts_iter = &mut accounts.iter();
74     let accounts = Accounts {
75         central_state_account: next_account_info(accounts_iter)?,
76         stake_account: next_account_info(accounts_iter)?,
77         stake_pool: next_account_info(accounts_iter)?,
78         owner: next_account_info(accounts_iter)?,
79         source_token: next_account_info(accounts_iter)?,
80         spl_token_program: next_account_info(accounts_iter)?,
81         vault: next_account_info(accounts_iter)?,
82         fee_account: next_account_info(accounts_iter)?,
83         bond_account: next_account_info(accounts_iter).ok(),
84     };
```

Listing 6: src/processor/stake.rs

```
109 check_account_owner(
110             accounts.source_token,
111             &spl_token::ID,
112             AccessError::WrongTokenAccountOwner,
113 )?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Although this does not imply a security risk since when the transfer is attempted, the transaction will fail if the above case occurs. However, it is recommended to add a check to verify that the owner account of the source_token account match the owner account provided.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit d4b1da28abfad5555815c296fd42d6fb347d7571: A check to verify if the source_token account matches the owner account was added to the Stake instruction handler.

3.3 (HAL-03) MINT OF DESTINATION ACCOUNT CHECK MISSING - INFORMATIONAL

Description:

The ClaimBondRewards instruction allows transferring the generated bond rewards to the rewards_destination account provided. If this account does not belong to the bond owner, they must sign the transaction.

The instruction handler does not check however if the mint of the rewards_destination account matches the mint of central state.

This also happens in other instruction handlers, claim_rewards and claim_pool_rewards. Additionally, this was observed in the unstake function and the destination_token account, in the stake function and the source_token account, and in the unlock_bond_tokensfunction and the access_token_destination account.

Code Location:

Listing 7: src/processor/claim_bond_rewards.rs (Line 45)

```
30 pub struct Accounts<'a, T> {
31     /// The stake pool account
32     #[cons(writable)]
33     pub stake_pool: &'a T,
34
35     /// The bond account
36     #[cons(writable)]
37     pub bond_account: &'a T,
38
39     /// The bond account owner
40     #[cons(signer)]
41     pub bond_owner: &'a T,
42
43     /// The rewards destination
44     #[cons(writable)]
```

```
45      pub rewards_destination: &'a T,
46
47      /// The central state account
48      pub central_state: &'a T,
```

Listing 8: src/processor/claim_bond_rewards.rs

```
92  check_account_owner(
93              accounts.rewards_destination,
94              &spl_token::ID,
95              AccessError::WrongOwner,
96          )?;
```

Listing 9: src/processor/claim_bond_rewards.rs (Line 167)

```
144 check_account_key(
145         accounts.mint,
146         &central_state.token_mint,
147         AccessError::WrongMint,
148     )?;
149
150     let reward = calc_reward_fp32(
151         central_state.last_snapshot_offset,
152         bond.last_claimed_offset,
153         &stake_pool,
154         true,
155         false,
156     )?
157     // Multiply by the staker shares of the total pool
158     .checked_mul(bond.total_staked as u128)
159     .map(|r| ((r >> 31) + 1) >> 1)
160     .and_then(safe_downcast)
161     .ok_or(AccessError::Overflow)?;
162
163     // Transfer rewards
164     let transfer_ix = mint_to(
165         &spl_token::ID,
166         accounts.mint.key,
167         accounts.rewards_destination.key,
168         accounts.central_state.key,
169         &[],
170         reward,
171     )?;
```

```
172     invoke_signed(
173         &transfer_ix,
174         &[
175             accounts.spl_token_program.clone(),
176             accounts.mint.clone(),
177             accounts.central_state.clone(),
178             accounts.rewards_destination.clone(),
179         ],
180         &[&[&program_id.to_bytes(), &[central_state.signer_nonce
 ↳ ]]],
181     )?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Although this does not introduce a security risk, since when attempting to perform the minting, the transaction fails if the above case occurs. However, it is recommended to add a check in the instruction handler to verity the mint of rewards_destination token account matches the mint of central state account.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 94221402c98e393eaab23bbe1e2084d44dc3964d: Checks were added to the affected instruction handlers to verify the mint of the destination account matches the mint of the central state account.

3.4 (HAL-04) MISLEADING ACCESS ERROR - INFORMATIONAL

Description:

The Stake and Unstake instructions allow the transaction sender to optionally include a bond account. This account is used to adjust the value of pool_minimum_at_creation of the stake account, as well as to calculate if the unstake amount is valid. If it is not, an **AccessError** WrongTokenAccountOwner is thrown. However, this error is not representative, since the bond account is not a token account.

Code Location:

Listing 10: src/processor/stake.rs (Line 125)

```
121 if let Some(bond_account) = accounts.bond_account {
122     check_account_owner(
123         bond_account,
124         program_id,
125         AccessError::WrongTokenAccountOwner,
126     )?
127 }
```

Listing 11: src/processor/unstake.rs (Line 111)

```
107 if let Some(bond_account) = accounts.bond_account {
108     check_account_owner(
109         bond_account,
110         program_id,
111         AccessError::WrongTokenAccountOwner,
112     )?
113 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to create a new AccessError that properly represents an incorrect bond account owner.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commits 94221402c98e393eaab23bbe1e2084d44dc3964d and a056460212b2261ad6c110e2c1801da2b86dfc2d: A new **AccessError**, called WrongBondAccountOwner, was introduced and added to the checks where the bond account provided is verified in the Stake and Unstake instructions handlers.

3.5 (HAL-05) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL

Description:

It was observed that there is no overflow-checks=true in Cargo.toml. By default, overflow checks are disabled in optimized release builds. Hence, if there is an overflow in release builds, it will be silenced, leading to unexpected behavior of an application. Even if checked arithmetic is used through checked_*, it is recommended to have that check in Cargo.toml.

Code Location:

- program/Cargo.toml

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add overflow-checks=true under your release profile in Cargo.toml.

Remediation Plan:

SOLVED: The Access Labs team fixed this issue in commit 50aa3ea87a550115694d019c4ebbe63bb16c0a7a: the overflow-checks=true property was added to the package manifest.

3.6 (HAL-06) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL

Description:

The use of helper methods in Rust, such as unwrap, is allowed in dev and testing environment because those methods are supposed to throw an error (also known as panic!) when called on Option::None or a Result which is not Ok. However, keeping unwrap functions in production environment is considered bad practice because they may lead to program crashes, which are usually accompanied by insufficient or misleading error messages.

Code Location:

Listing 12

```
./program/state.rs:151:                try_cast_slice_mut(rem).
↳ unwrap(),
./program/state.rs:450:        let current_time = Clock::get().
↳ unwrap().unix_timestamp as u64;
./program/state.rs:579:        let current_time = Clock::get().
↳ unwrap().unix_timestamp;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to use the unwrap function in the production environment because its use causes 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 unwrapping, or using the error-chain crate for errors.

Remediation Plan:

PARTIALLY SOLVED: The Access Labs team fixed this issue in commit 694e13afe6367c346414eada6af9df46df897538: The get_current_offset and activate functions in state.rs have been modified to replace unwraps with propagating errors with ?; all references to those functions have also been updated accordingly. Access Labs explained that modifying the get_checked function involves more complex code changes and because this finding does not pose a direct security risk, this function was not updated.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

4.1 UNSTAKE AN INVALID AMOUNT FROM STAKE POOL

Description:

The Unstake instruction allows stakers to unstake the amount they staked, provided they claimed their rewards. The possibility to provide the corresponding account bond has been added.

The stake account has a pool_minimum_at_creation field that represents the minimum stake amount of the stake pool at the time of its creation.

This field is updated when the Unstake instruction is executed for consistency with the minimum amount to stake to get access to the pool in the stake pool. However, if the amount to unstake is not the total amount staked in it, it could result in a number of tokens staked in that account below the minimum allowed.

Results:

```
[+] UnStake  -------> Instruction { program_id: acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX, accounts: [AccountMeta { pubkey: 7rA53R7EzECk9f3nmppKufkUZxXq9ptCRRwcRPvocuNL, is_signer: false, is_writable: t
rue }, AccountMeta { pubkey: 9mxJeTzinXHCBwm9YkseXEfJViMi3ueX86CvFs69cMF4s, is_signer: false, is_writable: true }, AccountMeta { pubkey: 3QKCmhsfCLMpTvkiADFV3xMsVurAQyirqzY7po44fKtv, is_signer: false, is_w
ritable: true }, AccountMeta { pubkey: F6qj64QptcWZVwvgJXsNfauyAGLFentss1pLuDnA3wes, is_signer: true, is_writable: false }, AccountMeta { pubkey: 11111113ih1vYV5YuKP6Ah5B6fbRdMw9XHiZAvAaj, is_signer: fals
e, is_writable: true }, AccountMeta { pubkey: TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA, is_signer: false, is_writable: false }, AccountMeta { pubkey: 1111111ogCyDbaRMvkdsHB3qfdyFYaG1WtRUAfdh, is_signer
: false, is_writable: true }, AccountMeta { pubkey: FDmt87mn45d8eY6mxP8TG8UiU9fiyfyt14xdCp6edZRt, is_signer: false, is_writable: false }], data: [5, 58, 7, 0, 0, 0, 0, 0, 0] }
amount to unstake : 1850
[2023-01-20T07:22:35.450721000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX invoke [1]
[2023-01-20T07:22:35.451367000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Entrypoint
[2023-01-20T07:22:35.451396000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Beginning processing
[2023-01-20T07:22:35.451416000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction unpacked
[2023-01-20T07:22:35.451431000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Unstake
[2023-01-20T07:22:35.452847000Z DEBUG solana_runtime::message_processor::stable_log] Program log: totak staked in bond: 1000
[2023-01-20T07:22:35.453070000Z DEBUG solana_runtime::message_processor::stable_log] Program log: total staked amount in stake account: 1200
[2023-01-20T07:22:35.453286000Z DEBUG solana_runtime::message_processor::stable_log] Program log: new_total_in_pool: 350
[2023-01-20T07:22:35.453331000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Error: Invalid unstake amount
[2023-01-20T07:22:35.453534000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX consumed 7308 of 200000 compute units
[2023-01-20T07:22:35.453592000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX failed: custom program error: 0x21
thread 'poc' panicked at 'called `Result::unwrap()` on an `Err` value: TransactionError(InstructionError(0, Custom(33)))', tests/security_access.rs:1151:45
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test poc ... FAILED

failures:

failures:
```

No code vulnerabilities were identified.

4.2 TESTING CHANGES IN THE TIME FIELDS OF ACCOUNTS

Some changes have been added to the account fields to keep track of the time elapsed between actions like claim, stake, etc.
A function has also been added, `get_current_offset()`, to make it easier to calculate the number of days given the values of these fields. These were tested to confirm no new vulnerabilities were introduced.

Results:

```
Calling Stake timestamp when calling Stake : 1674207110
[2023-01-20T09:31:50.900386000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX invoke [1]
[2023-01-20T09:31:50.901060000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Entrypoint
[2023-01-20T09:31:50.901085000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Beginning processing
[2023-01-20T09:31:50.901105000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction unpacked
[2023-01-20T09:31:50.901119000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Stake
[2023-01-20T09:31:50.909005000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2023-01-20T09:31:50.909392000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2023-01-20T09:31:50.910808000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 190815 compute units
[2023-01-20T09:31:50.910868000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2023-01-20T09:31:50.917194000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2023-01-20T09:31:50.917575000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2023-01-20T09:31:50.918965000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 183158 compute units
[2023-01-20T09:31:50.919022000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2023-01-20T09:31:50.919692000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX consumed 22876 of 200000 compute units
[2023-01-20T09:31:50.919763000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX success
[+] Stake  done!
[+] Stake
Calling Stake timestamp when calling Stake : 1674507110
[2023-01-20T09:31:50.993475000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX invoke [1]
[2023-01-20T09:31:50.994161000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Entrypoint
[2023-01-20T09:31:50.994188000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Beginning processing
[2023-01-20T09:31:50.994208000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction unpacked
[2023-01-20T09:31:50.994224000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Stake
[2023-01-20T09:31:50.995802000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Error: Pool must be cranked
```

No code vulnerabilities were identified.

4.3 ACCESS CONTROL IN CLAIM BOND

In ClaimBond the possibility has been introduced for the instruction to be permissionless if the bond account was created with a non-zero quote amount.
This was tested to confirm no vulnerabilities were introduced with this change.

Results:

```
[+] ClaimBond
[2023-01-20T10:14:39.327159000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX invoke [1]
[2023-01-20T10:14:39.327841000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Entrypoint
[2023-01-20T10:14:39.327867000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Beginning processing
[2023-01-20T10:14:39.327897000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction unpacked
[2023-01-20T10:14:39.327913000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: claim bond
[2023-01-20T10:14:39.332640000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Bond owner: EewUbg6zHFK46uikdTcUKyq1sW15hGstQwg23khB61Bf
[2023-01-20T10:14:39.336289000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Buyer provided: 9ZGuzmv2FoLrMb8Fz2c49fZtMq6t33icfzcNvcZ78ST
[2023-01-20T10:14:39.336529000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Bond total quote amount: 8999999900000
[2023-01-20T10:14:39.336715000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Error: Account not generated deterministically
[2023-01-20T10:14:39.336961000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX consumed 30530 of 200000 compute units
[2023-01-20T10:14:39.337012000Z DEBUG solana_runtime::message_processor::stable_log] Program acp1VPqNoMs5KC5aEH3MzxnyPZNyKQF1TCPouCoNRuX failed: custom program error: 0x5
```

No code vulnerabilities were identified.

AUTOMATED TESTING

5.1 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 found two unsafe arithmetic operations, which one of them may result in overflow. It was reported with HAL-01 vulnerability in previous chapter. The rest of issues identified were verified to be false positives.

```
 - ✓ [00m:02s] Loading IR From File
 - ▪ [00m:00s] Running Compiler Optimization Passes
EntryPoints:
entrypoint
 - ✓ [00m:00s] Running Compiler Optimization Passes
 - ✓ [00m:01s] Running Pointer Analysis
=============This account may be UNTRUSTFUL!================
Found a potential vulnerability at line 120, column 50 in src/processor/claim_bond_rewards.rs
The account info is not trustful:

 114|    let accounts = Accounts::parse(accounts, program_id)?;
 115|
 116|    let central_state = CentralState::from_account_info(accounts.central_state)?;
 117|    let stake_pool = StakePool::get_checked(accounts.stake_pool, vec![Tag::StakePool])?;
 118|    let mut bond = BondAccount::from_account_info(accounts.bond_account, false)?;
 119|
>120|    let destination_token_acc = Account::unpack(&accounts.rewards_destination.data.borrow())?;
 121|
 122|    if destination_token_acc.owner != bond.owner {
 123|        // If the destination does not belong to the bond owner he must sign
 124|        check_signer(accounts.bond_owner, AccessError::BuyerMustSign)?;
 125|    } else {
 126|        assert_no_close_or_delegate(&destination_token_acc)?;
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h6f073fd945cde336 [src/entrypoint.rs:21]
>>>  access_protocol::processor::claim_bond_rewards::process_claim_bond_rewards::hf9078022f045be73 [src/processor.rs:144]
```

AUTOMATED TESTING

```
=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 180, column 8 in src/processor/stake.rs
The add operation may result in overflows:

 174|        )?;
 175|
 176|        amount_in_bonds = bond_account.total_staked;
 177|    }
 178|
 179|    // if we were previously under the minimum stake limit it gets reset to the pool's one
>180|    if stake_account.stake_amount + amount_in_bonds < stake_account.pool_minimum_at_creation {
 181|        stake_account.pool_minimum_at_creation = stake_pool.header.minimum_stake_amount;
 182|    }
 183|
 184|
 185|    assert_valid_fee(accounts.fee_account, &central_state.authority)?;
 186|
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h6f073fd945cde336 [src/entrypoint.rs:21]
>>>  access_protocol::processor::stake::process_stake::h28bf3aa7d7107ef0 [src/processor.rs:74]
```

```
=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 185, column 30 in src/state.rs
The sub operation may result in underflows:

 179|    pub fn push_balances_buff(
 180|        &mut self,
 181|        current_offset: u64,
 182|        last_crank_offset: u64,
 183|        rewards: RewardsTuple,
 184|    ) -> Result<(), ProgramError> {
>185|        let nb_days_passed = current_offset - last_crank_offset;
 186|        for i in 1..nb_days_passed {
 187|            self.balances[(((self.header.current_day_idx as u64)
 188|                .checked_add(i)
 189|                .ok_or(AccessError::Overflow)?)
 190|                % STAKE_BUFFER_LEN) as usize] = RewardsTuple {
 191|                pool_reward: 0,
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h6f073fd945cde336 [src/entrypoint.rs:21]
>>>  access_protocol::processor::crank::process_crank::h4f3b3d919c42ed77 [src/processor.rs:98]
>>>    access_protocol::state::StakePool$LT$H$C$B$GT$::push_balances_buff::hb468c6b2f02c9eea [src/processor/crank.rs:120]

=============This account may be UNTRUSTFUL!================
Found a potential vulnerability at line 117, column 50 in src/processor/claim_rewards.rs
The account info is not trustful:

 111|    let accounts = Accounts::parse(accounts, program_id)?;
 112|
 113|    let central_state = CentralState::from_account_info(accounts.central_state)?;
 114|    let stake_pool = StakePool::get_checked(accounts.stake_pool, vec![Tag::StakePool])?;
 115|    let mut stake_account = StakeAccount::from_account_info(accounts.stake_account)?;
 116|
>117|    let destination_token_acc = Account::unpack(&accounts.rewards_destination.data.borrow())?;
 118|    msg!("Account owner: {}", destination_token_acc.owner);
 119|
 120|    if destination_token_acc.owner != stake_account.owner {
 121|        // If the destination does not belong to the staker he must sign
 122|        check_signer(accounts.owner, AccessError::StakePoolOwnerMustSign)?;
 123|    } else {
>>>Stack Trace:
>>>access_protocol::processor::Processor::process_instruction::h6f073fd945cde336 [src/entrypoint.rs:21]
```

5.2 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.

Results:

ID	package	Short Description
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations.
RUSTSEC-2020-0071	time	Potential segfault in the time crate.
RUSTSEC-2023-00011	tokio	reject_remote_clients Configuration corruption.

5.3 UNSAFE RUST CODE DETECTION

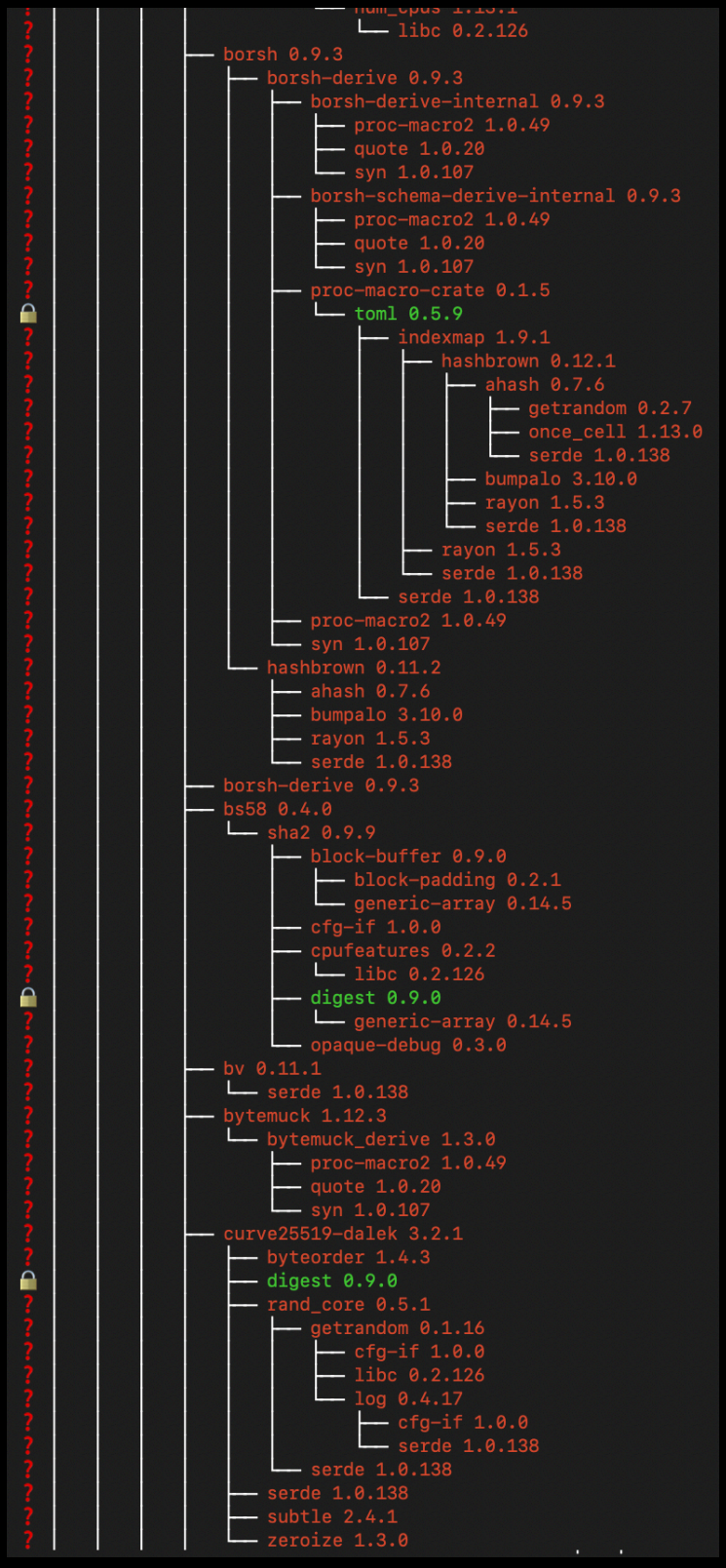
Description:

Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

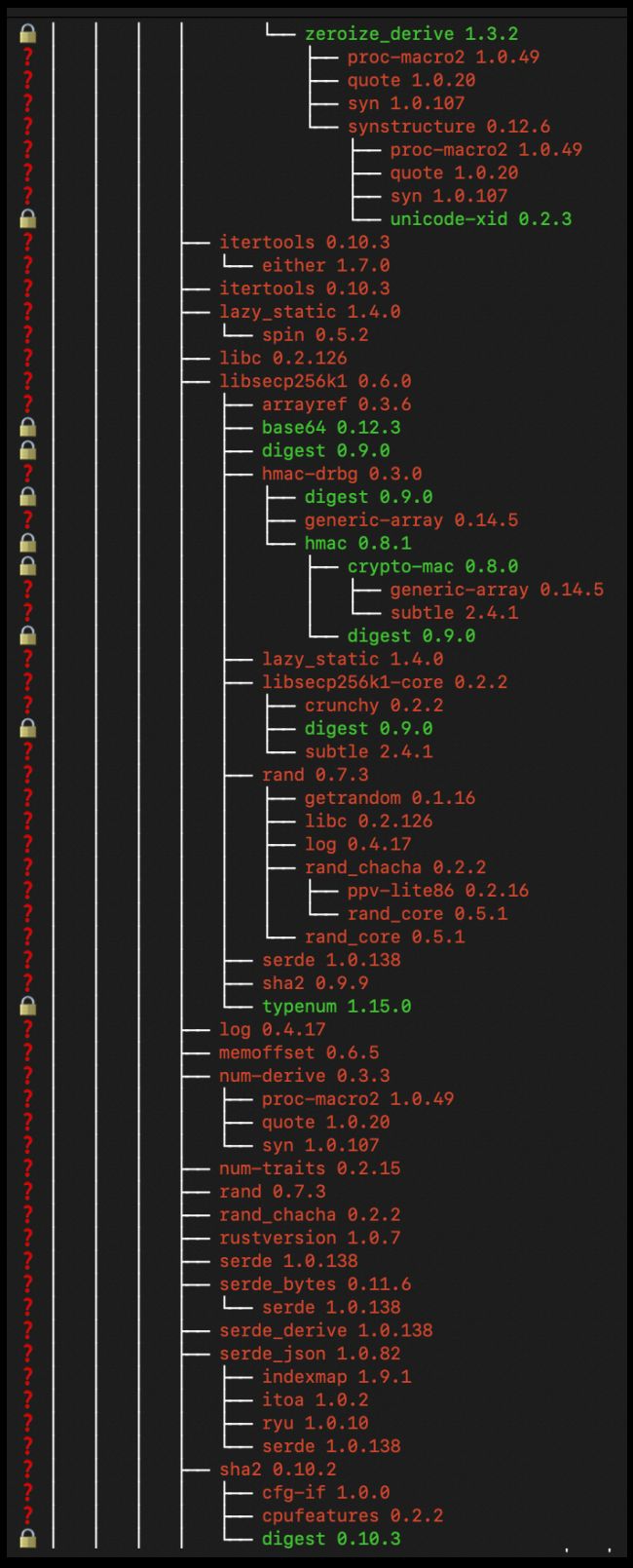
Results:


Symbols:
🔒 = All entry point .rs files declare #![forbid(unsafe_code)].
❓ = This crate may use unsafe code.
❓ access-protocol 0.1.0
❓ ├── bonfida-utils 0.2.12
❓ │ ├── bonfida-macros 0.2.3
❓ │ │ ├── proc-macro2 1.0.49
❓ │ │ │ └── unicode-ident 1.0.1
❓ │ │ ├── quote 1.0.20
❓ │ │ │ └── proc-macro2 1.0.49
❓ │ │ ├── solana-program 1.14.12
🔒 │ │ │ ├── base64 0.13.0
❓ │ │ │ ├── bincode 1.3.3
❓ │ │ │ │ └── serde 1.0.138
❓ │ │ │ │ └── serde_derive 1.0.138
❓ │ │ │ │ ├── proc-macro2 1.0.49
❓ │ │ │ │ ├── quote 1.0.20
❓ │ │ │ │ └── syn 1.0.107
❓ │ │ │ │ ├── proc-macro2 1.0.49
❓ │ │ │ │ ├── quote 1.0.20
❓ │ │ │ │ └── unicode-ident 1.0.1
❓ │ │ │ ├── bitflags 1.3.2
❓ │ │ │ ├── blake3 1.3.1
❓ │ │ │ │ ├── arrayref 0.3.6
❓ │ │ │ │ ├── arrayvec 0.7.2
❓ │ │ │ │ │ └── serde 1.0.138
❓ │ │ │ │ ├── cfg-if 1.0.0
❓ │ │ │ │ ├── constant_time_eq 0.1.5
🔒 │ │ │ │ ├── digest 0.10.3
❓ │ │ │ │ │ ├── block-buffer 0.10.2
❓ │ │ │ │ │ │ └── generic-array 0.14.5
❓ │ │ │ │ │ │ ├── serde 1.0.138
🔒 │ │ │ │ │ │ └── typenum 1.15.0
🔒 │ │ │ │ │ ├── crypto-common 0.1.4
❓ │ │ │ │ │ │ ├── generic-array 0.14.5
❓ │ │ │ │ │ │ └── rand_core 0.6.3
❓ │ │ │ │ │ │ ├── getrandom 0.2.7
❓ │ │ │ │ │ │ │ ├── cfg-if 1.0.0
❓ │ │ │ │ │ │ │ └── libc 0.2.126
❓ │ │ │ │ │ │ └── serde 1.0.138
🔒 │ │ │ │ │ └── typenum 1.15.0
❓ │ │ │ │ └── subtle 2.4.1
❓ │ │ │ └── rayon 1.5.3
❓ │ │ │ ├── crossbeam-deque 0.8.1
❓ │ │ │ │ ├── cfg-if 1.0.0
❓ │ │ │ │ ├── crossbeam-epoch 0.9.9
❓ │ │ │ │ │ ├── cfg-if 1.0.0
❓ │ │ │ │ │ ├── crossbeam-utils 0.8.10
❓ │ │ │ │ │ │ ├── cfg-if 1.0.0
❓ │ │ │ │ │ │ └── once_cell 1.13.0
❓ │ │ │ │ │ │ └── parking_lot_core 0.9.3
❓ │ │ │ │ │ │ ├── cfg-if 1.0.0
❓ │ │ │ │ │ │ ├── libc 0.2.126
❓ │ │ │ │ │ │ └── smallvec 1.9.0
❓ │ │ │ │ │ │ └── serde 1.0.138
❓ │ │ │ │ │ ├── memoffset 0.6.5
❓ │ │ │ │ │ ├── once_cell 1.13.0
❓ │ │ │ │ │ └── scopeguard 1.1.0
❓ │ │ │ │ └── crossbeam-utils 0.8.10
❓ │ │ │ ├── either 1.7.0
❓ │ │ │ │ └── serde 1.0.138
❓ │ │ │ └── rayon-core 1.9.3
❓ │ │ │ ├── crossbeam-channel 0.5.5
❓ │ │ │ │ ├── cfg-if 1.0.0

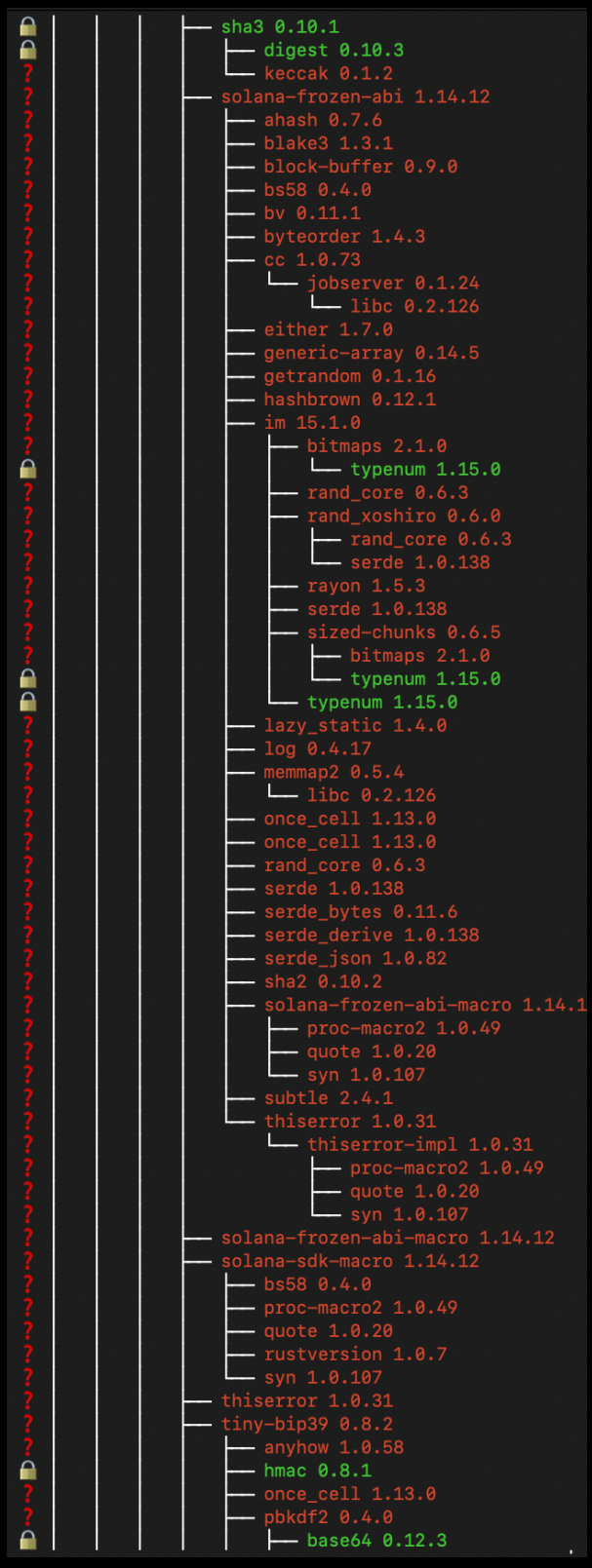
AUTOMATED TESTING



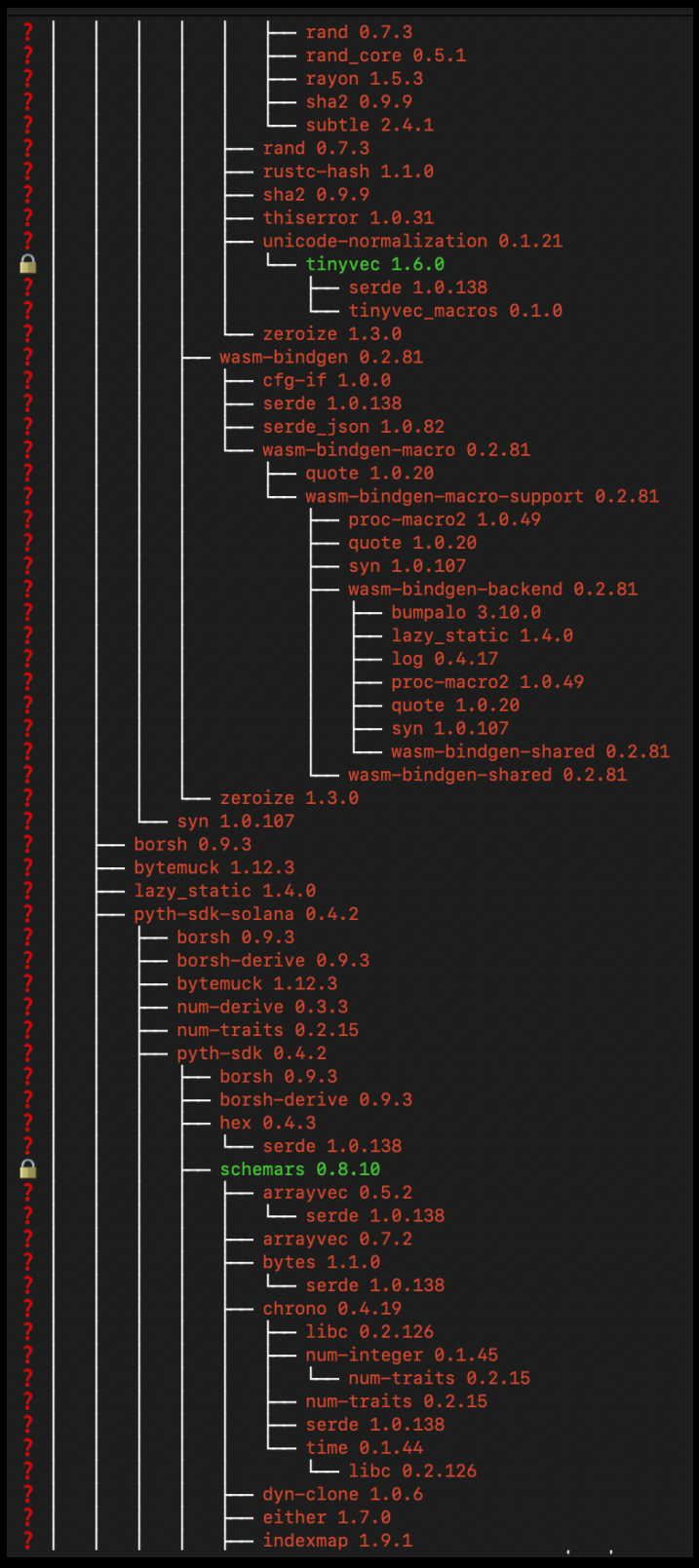
AUTOMATED TESTING



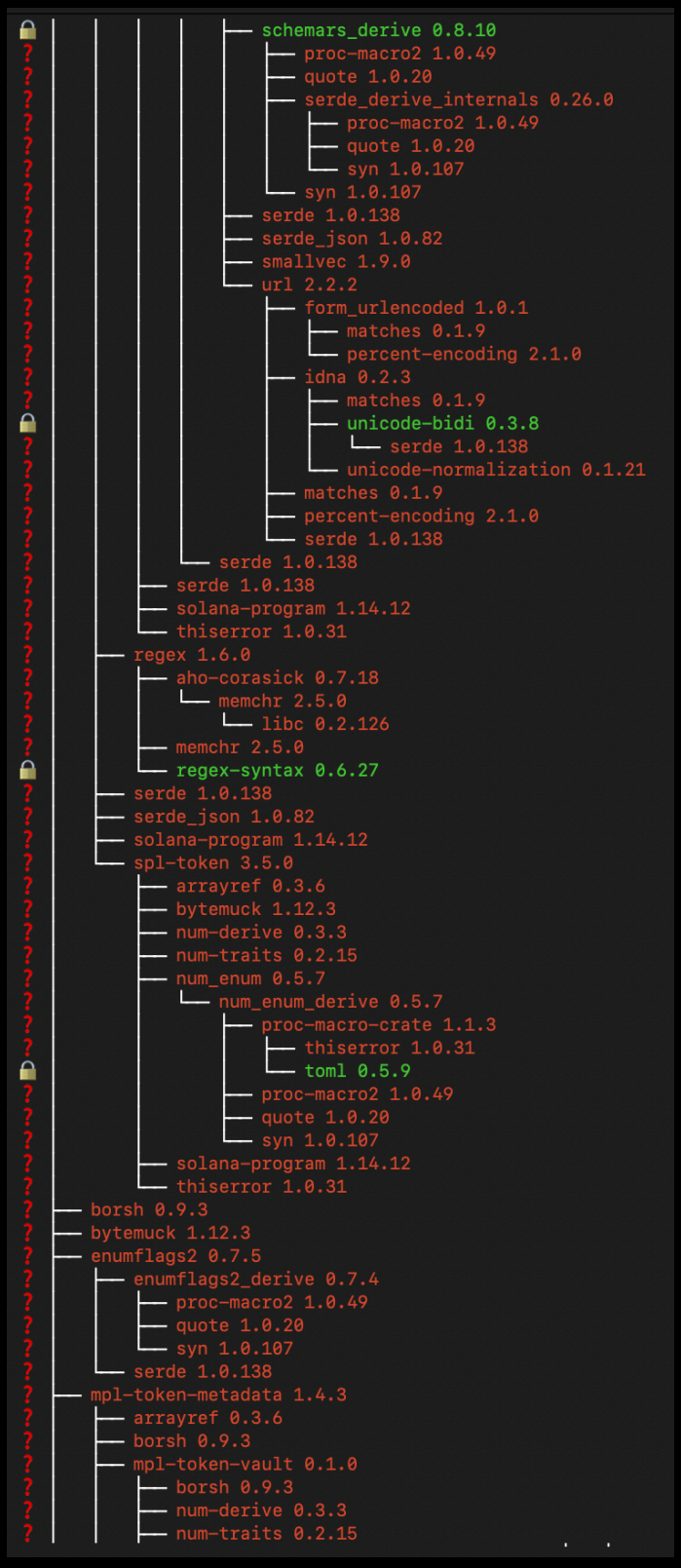
AUTOMATED TESTING



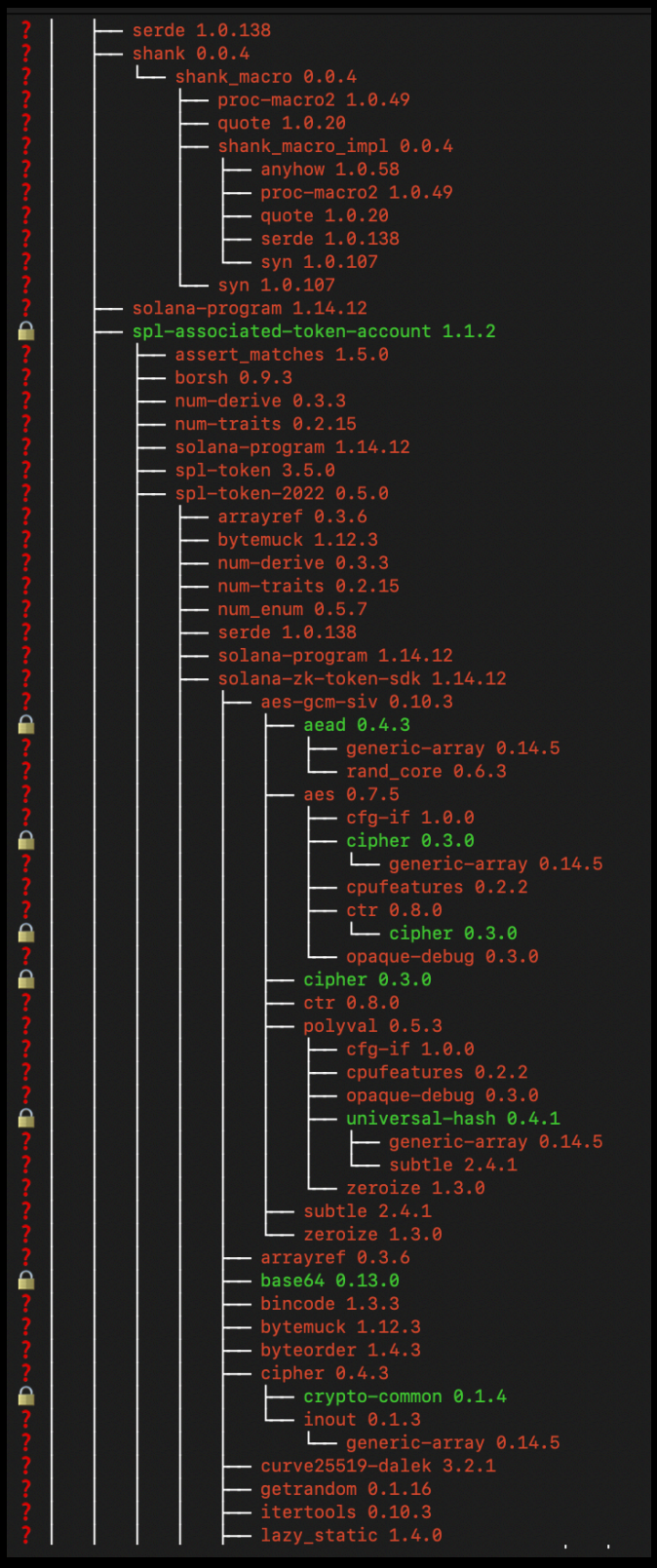
AUTOMATED TESTING



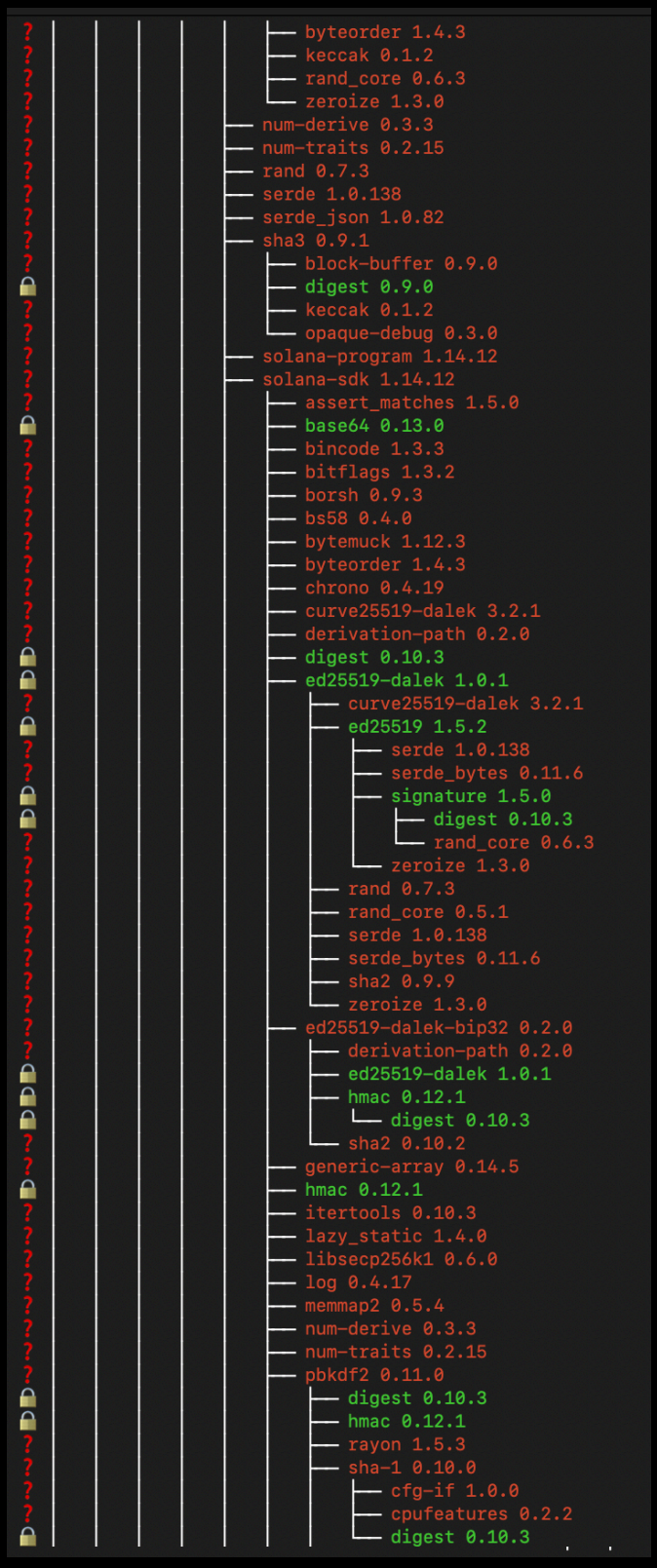
AUTOMATED TESTING



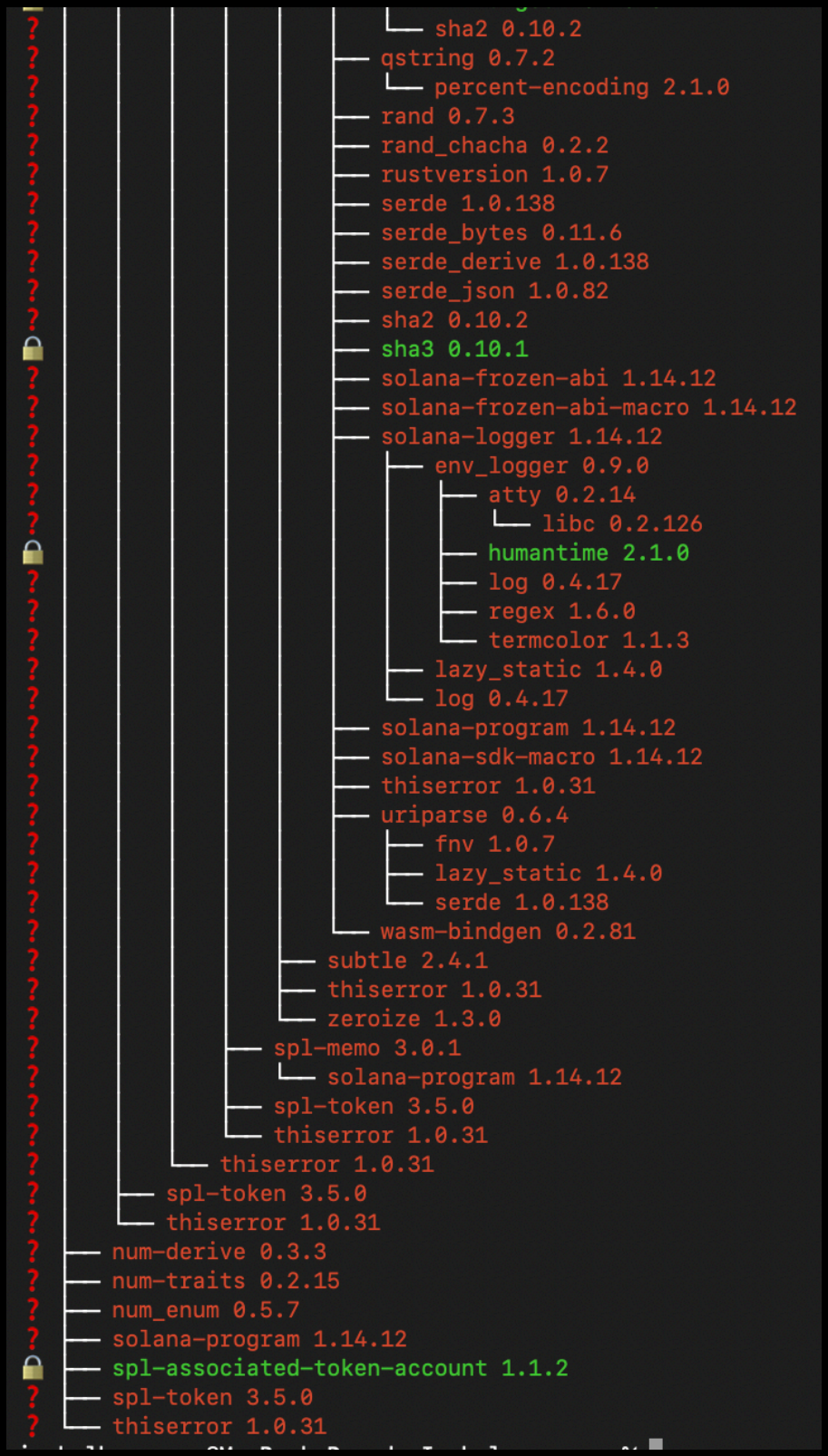
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN