// HALBORN

Solana Labs - Solana Update v1.11.3

L1 Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 21st, 2022 - August 19th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/01/2022	Isabel Burruezo
0.2	Draft Review	09/07/2022	Gabi Urrutia
0.3	Draft Edits	09/15/2022	Isabel Burruezo
0.4	Draft Review	09/15/2022	Piotr Cielas
0.5	Draft Review	09/15/2022	Gabi Urrutia
1.0	Remediation Plan	05/19/2023	Guillermo Alvarez
1.1	Remediation Plan Review	05/19/2023	Isabel Burruezo
1.2	Remediation Plan Review	05/19/2023	Piotr Cielas
1.3	Remediation Plan Review	05/19/2023	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
Guillermo Álvarez	Halborn	Guillermo.Alvarez@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Sealevel, Solana's parallel smart contracts runtime, is able to process transactions in parallel because Solana transactions describe all the states a transaction reads or writes to while being processed. This not only allows for non-overlapping transactions to execute concurrently, but also for transactions that are only reading the same state to execute concurrently as well.

Halborn conducted a security audit on the Sealevel runtime on July 21st, 2022 and ending on August 19th, 2022 . The security assessment was scoped to the implementation of the update of the runtime provided in the solana 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 smart contract 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 did not identify any security risk affecting the Updated version of the Runtime.

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 this 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 protocol.
- Manual code review and walkthrough to identify possible logic issues.
- 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)

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

Code repositories:

1. runtime

- Repository: solana
- Release Tags:
  - v1.11.3
- Packages in scope:
  1. runtime (runtime)

Solana Labs advised Halborn to center the runtime audit around the solana_runtime::Bank::load_and_execute_transaction function in the solana-runtime package.

Out-of-scope:

- accountsdb.rs
- versioned transactions and accounts lookup table (in scope of another scheduled audit)
- External libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	4

LIKELIHOOD

IMPACT

(HAL-02)				
			(HAL-01)	
(HAL-03)				
(HAL-04) (HAL-05) (HAL-06)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SOME BUILT-IN PROGRAMS DO NOT CONSUME COMPUTE UNITS	Medium	SOLVED - 03/24/2023
(HAL-02) TRANSACTION PRIORITIZATION IS NOT ENFORCED	Low	RISK ACCEPTED
(HAL-03) CHANGING LAMPORTS PER SIGNATURE MIGHT NOT AFFECT TOTAL FEES	Informational	SOLVED - 02/01/2023
(HAL-04) CHECKED ARITHMETIC MISSING	Informational	ACKNOWLEDGED
(HAL-05) EXTRA ALLOCATION ON VECTOR RESIZE MAY AFFECT CLUSTER PERFORMANCE	Informational	ACKNOWLEDGED
(HAL-06) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) SOME BUILT-IN PROGRAMS DO NOT CONSUME COMPUTE UNITS - MEDIUM

Description:

In Solana, work performed by the cluster while processing transactions is measured in Compute Units. The total number of Compute Units transactions require is calculated with the `get_transaction_cost` function. For built-in programs, the cost of processing their instructions is static and defined in the runtime.

However, the `BPFLoaderUpgradeab1e11111111111111111111111` (responsible for deploying, upgrading, and executing programs) and `Ed25519SigVerify111111111111111111111111111` (responsible for verifying ed25519 signatures) native programs are not included in the `BUILT_IN_INSTRUCTION_COSTS` HashMap.

Because both those programs are essential to the runtime and, as such, are invoked very regularly, whenever they are called, transaction cost is underestimated and the cluster ends up performing more work than it is allowed to.

Code Location:

Listing 1: runtime/src/block_cost_limits.rs

```
32 /// Number of compute units for each built-in programs
33 pub static ref BUILT_IN_INSTRUCTION_COSTS: HashMap<Pubkey, u64> =
↳ [
34     (feature::id(), COMPUTE_UNIT_TO_US_RATIO * 2),
35     (incinerator::id(), COMPUTE_UNIT_TO_US_RATIO * 2),
36     (native_loader::id(), COMPUTE_UNIT_TO_US_RATIO * 2),
37     (solana_sdk::stake::config::id(), COMPUTE_UNIT_TO_US_RATIO *
↳ 2),
38     (solana_sdk::stake::program::id(), COMPUTE_UNIT_TO_US_RATIO *
↳ 25),
```

```
39      (solana_config_program::id(), COMPUTE_UNIT_TO_US_RATIO * 15),
40      (solana_vote_program::id(), COMPUTE_UNIT_TO_US_RATIO * 70),
41      // secp256k1 is executed in banking stage, it should cost
↳ similar to sigverify
42      (secp256k1_program::id(), COMPUTE_UNIT_TO_US_RATIO * 24),
43      (system_program::id(), COMPUTE_UNIT_TO_US_RATIO * 5),
44  ]
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to include BPFLoaderUpgradeab1e11111111111111111111 and Ed25519SigVerify111111111111111111111111111 in the BUILT_IN_INSTRUCTION_COSTS HashMap and assign them appropriate amounts of Compute Units to ensure that transaction costs are estimated accurately.

Remediation Plan:

SOLVED: The Solana team solved the issue in pull request 30702. All built-in programs now are required to consume compute units. A new error BuiltinProgramsMustConsumeComputeUnits was implemented in the native instruction processor to ensure that if the feature native_programs_consume_cu is activated, this issue does not occur again in the future.

3.2 (HAL-02) TRANSACTION PRIORITIZATION IS NOT ENFORCED - LOW

Description:

The ComputeBudgetInstruction::SetComputeUnitPrice instruction is used to set a custom compute unit price in micro-lamports for higher transaction prioritization. Thus, the block producer's scheduling algorithm tries to include high prioritization transactions first in a block.
This amount of micro-lamports is added as prioritization_fee to the total fee calculated with the calculate_fee function.

Whenever this instruction is found in a transaction, the transaction payer pays the prioritization_fee on top of other fees, however the **validator** can decide whether to prioritize it and include it in the block or not, since this fee is only an incentive. In case the **validator** decides not to prioritize such a transaction, the payer loses that prioritization_fee.

Code Location:

Listing 2: runtime/src/bank.rs (Lines 4787,4788,4794)

```
4771 pub fn calculate_fee(
4772         message: &SanitizedMessage,
4773         lamports_per_signature: u64,
4774         fee_structure: &FeeStructure,
4775         support_set_compute_unit_price_ix: bool,
4776     ) -> u64 {
4777         // Fee based on compute units and signatures
4778         const BASE_CONGESTION: f64 = 5_000.0;
4779         let current_congestion = BASE_CONGESTION.max(
  ↳ lamports_per_signature as f64);
4780         let congestion_multiplier = if lamports_per_signature == 0
  ↳  {
4781             0.0 // test only
4782         } else {
4783             BASE_CONGESTION / current_congestion
```

```
4784         };
4785
4786         let mut compute_budget = ComputeBudget::default();
4787         let prioritization_fee_details = compute_budget
4788             .process_instructions(
4789                 message.program_instructions_iter(),
4790                 false,
4791                 support_set_compute_unit_price_ix,
4792             )
4793             .unwrap_or_default();
4794         let prioritization_fee = prioritization_fee_details.
  ↳ get_fee();
```

Listing 3: runtime/src/bank.rs (Line 4493)

```
4485 let compute_budget = if let Some(compute_budget) =  self.
  ↳ compute_budget {
4486             compute_budget
4487         } else {
4488             let mut compute_budget =
4489                 ComputeBudget::new(compute_budget::
  ↳ MAX_COMPUTE_UNIT_LIMIT as u64);
4490
4491             let mut compute_budget_process_transaction_time =
4492                 Measure::start("
  ↳ compute_budget_process_transaction_time");
4493             let process_transaction_result = compute_budget.
  ↳ process_instructions(
4494                 tx.message().program_instructions_iter(),
4495                 feature_set.is_active(&
  ↳ default_units_per_instruction::id()),
4496                 feature_set.is_active(&
  ↳ add_set_compute_unit_price_ix::id()),
4497             );
```

Listing 4: program-runtime/src/compute_budget.rs

```
174 Ok(ComputeBudgetInstruction::SetComputeUnitPrice(micro_lamports))
  ↳ => {
175     if prioritization_fee.is_some() {
176         return Err(duplicate_instruction_error);
177     }
178     prioritization_fee =
```

```
179         Some(PrioritizationFeeType::ComputeUnitPrice(
 ↳ micro_lamports));
180 }
181 _ => return Err(invalid_instruction_data_error),
```

Listing 5: program-runtime/src/priorization_fee.rs

```
32 PrioritizationFeeType::ComputeUnitPrice(cu_price) => {
33     let fee = {
34         let micro_lamport_fee: MicroLamports =
35             (cu_price as u128).saturating_mul(compute_unit_limit
 ↳ as u128);
36         let fee = micro_lamport_fee
37             .saturating_add(MICRO_LAMPORTS_PER_LAMPORT.
 ↳ saturating_sub(1) as u128)
38             .saturating_div(MICRO_LAMPORTS_PER_LAMPORT as u128);
39         u64::try_from(fee).unwrap_or(u64::MAX)
40     };
41
42     Self {
43         fee,
44         priority: cu_price,
45     }
46 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to force **validators** to return the prioritization fees to the payer in case the transaction is not considered over others and included in the block. One solution would be to calculate the total transaction fee with and without the prioritization fee and refund the delta to the payer if the transaction is not included in the current block.

Remediation Plan:

RISK ACCEPTED: The Solana team accepted the risk of this issue.

3.3 (HAL-03) CHANGING LAMPORTS PER SIGNATURE MIGHT NOT AFFECT TOTAL FEES - INFORMATIONAL

Description:

The calculate_fee function called by load_accounts is used to calculate the total fee to be paid by the transaction sender. One of the variables considered by the calculate_fee function is the congestion_multiplier parameter. The value of this parameter is calculated by dividing the constant BASE_CONGESTION hardcoded to 5000 by current_congestion. The value of the latter is either BASE_CONGESTION or lamports_per_signature, depending on which one is greater.

The possibility of changing the signature cost based on current cluster processing load indicated in FeeRateGovernor is still discussed.

When this is implemented, however, if lamports_per_signature increases due to higher cluster processing load, the value of congestion_multiplier will be lesser than 1. This would reduce the final signature fee to default (regardless of the cluster processing load) and the cluster would not be compensated appropriately based on the adjustment in the signature cost.

Code Location:

Listing 6: sdk/program/src/fee_calculator.rs (Lines 53-54,56)

```
52 pub struct FeeRateGovernor {
53     // The current cost of a signature  This amount may increase/
↳ decrease over time based on
54     // cluster processing load.
55     #[serde(skip)]
56     pub lamports_per_signature: u64,
57
58     // The target cost of a signature when the cluster is
↳ operating around target_signatures_per_slot
```

```
59      // signatures
60      pub target_lamports_per_signature: u64,
61
62      // Used to estimate the desired processing capacity of the
↳ cluster.  As the signatures for
63      // recent slots are fewer/greater than this value,
↳ lamports_per_signature will decrease/increase
64      // for the next slot.  A value of 0 disables
↳ lamports_per_signature fee adjustments
65      pub target_signatures_per_slot: u64,
66
67      pub min_lamports_per_signature: u64,
68      pub max_lamports_per_signature: u64,
69
70      // What portion of collected fees are to be destroyed, as a
↳ fraction of std::u8::MAX
71      pub burn_percent: u8,
72  }
```

Listing 7: sdk/program/src/fee_calculator.rs (Lines 12,14,16)

```
11  pub struct FeeCalculator {
12      /// The current cost of a signature.
13      ///
14      /// This amount may increase/decrease over time based on
↳ cluster processing
15      /// load.
16      pub lamports_per_signature: u64,
17  }
```

Listing 8: runtime/src/bank.rs (Lines 4783,4794-4795,4810-4815)

```
4770 /// Calculate fee for `SanitizedMessage`
4771 pub fn calculate_fee(
4772      message: &SanitizedMessage,
4773      lamports_per_signature: u64,
4774      fee_structure: &FeeStructure,
4775      support_set_compute_unit_price_ix: bool,
4776  ) -> u64 {
4777      // Fee based on compute units and signatures
4778      const BASE_CONGESTION: f64 = 5_000.0;
4779      let current_congestion = BASE_CONGESTION.max(
 ↳ lamports_per_signature as f64);
```

```
4780     let congestion_multiplier = if lamports_per_signature == 0 {
4781         0.0 // test only
4782     } else {
4783         BASE_CONGESTION / current_congestion
4784     };
4785     let mut compute_budget = ComputeBudget::default();
4786     let prioritization_fee_details = compute_budget
4787         .process_instructions(
4788             message.program_instructions_iter(),
4789             false,
4790             support_set_compute_unit_price_ix,
4791         )
4792         .unwrap_or_default();
4793     let prioritization_fee = prioritization_fee_details.get_fee();
4794     let signature_fee = Self::get_num_signatures_in_message(
  ↳ message)
4795         .saturating_mul(fee_structure.lamports_per_signature);
4796     let write_lock_fee = Self::get_num_write_locks_in_message(
  ↳ message)
4797         .saturating_mul(fee_structure.lamports_per_write_lock);
4798     let compute_fee = fee_structure
4799         .compute_fee_bins
4800         .iter()
4801         .find(|bin| compute_budget.compute_unit_limit <= bin.limit
  ↳ )
4802         .map(|bin| bin.fee)
4803         .unwrap_or_else(|| {
4804             fee_structure
4805                 .compute_fee_bins
4806                 .last()
4807                 .map(|bin| bin.fee)
4808                 .unwrap_or_default()
4809         });
4810     ((prioritization_fee
4811         .saturating_add(signature_fee)
4812         .saturating_add(write_lock_fee)
4813         .saturating_add(compute_fee) as f64)
4814         * congestion_multiplier)
4815         .round() as u64
4816 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

If this feature is added, it is recommended to add a check to verify if the new lamports_per_signature is greater than the hardcoded value of BASE CONGESTION. If it is, the formula for congestion_multiplier should be inverted. That is, by dividing lamports_per_signature by BASE CONGESTION instead.

Remediation Plan:

SOLVED: The Solana team fixed this issue in pull request 29828 by introducing a feature gate that removed the congestion_multiplier.

3.4 (HAL-04) CHECKED ARITHMETIC MISSING - INFORMATIONAL

Description:

Unsafe arithmetic operations (including multiplication, division, and addition) were identified in multiple files and program functions.

Code Location:

Listing 9: runtime/src/bank.rs (Lines 14,21)

```
pub fn calculate_fee(
        message: &SanitizedMessage,
        lamports_per_signature: u64,
        fee_structure: &FeeStructure,
        support_set_compute_unit_price_ix: bool,
    ) -> u64 {
        // Fee based on compute units and signatures
        const BASE_CONGESTION: f64 = 5_000.0;
        //el current congestion ser'a el max de lamports?per?
↳ signature
        let current_congestion = BASE_CONGESTION.max(
↳ lamports_per_signature as f64);
        let congestion_multiplier = if lamports_per_signature == 0
↳  {
            0.0 // test only
        } else {
            BASE_CONGESTION / current_congestion
        };
...
((prioritization_fee
            .saturating_add(signature_fee)
            .saturating_add(write_lock_fee)
            .saturating_add(compute_fee) as f64)
            * congestion_multiplier)
            .round() as u64
    }
```

Listing 10: runtime/src/accounts.rs

```rust
fn validate_fee_payer(
        payer_address: &Pubkey,
        payer_account: &mut AccountSharedData,
        payer_index: usize,
        error_counters: &mut TransactionErrorMetrics,
        rent_collector: &RentCollector,
        feature_set: &FeatureSet,
        fee: u64,
    ) -> Result<()> {
        if payer_account.lamports() == 0 {
            error_counters.account_not_found += 1;
            return Err(TransactionError::AccountNotFound);
        }
        let min_balance = match get_system_account_kind(
↳ payer_account).ok_or_else(|| {
            error_counters.invalid_account_for_fee += 1;
            TransactionError::InvalidAccountForFee
        })? {
            SystemAccountKind::System => 0,
            SystemAccountKind::Nonce => {
                rent_collector.rent.minimum_balance(NonceState::
↳ size())
            }
        };
        if payer_account.lamports() < fee + min_balance {
            error_counters.insufficient_funds += 1;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation Plan:

ACKNOWLEDGED: The Solana team acknowledged this issue.

3.5 (HAL-05) EXTRA ALLOCATION ON VECTOR RESIZE MAY AFFECT CLUSTER PERFORMANCE - INFORMATIONAL

Description:

The maximum number of accounts that can be referenced and locked in each transaction in the batch is defined with MAX_TX_ACCOUNT_LOCKS and set to 64.

load_transactions calls load_transaction on each transaction in the batch. All locked accounts are pushed to the accounts vector, and the program data accounts corresponding to each updatable program referenced are pushed to the accounts_deps vector. Both vectors are initialized with a capacity of 64 which can be extended up to 128 after the annexation of accounts_deps to accounts. Subsequently, load_executable_accounts is called to load the programs and add them to accounts.

If a transaction contains two instructions from two different upgradable programs and 60 upgradable programs are referenced as read-only, the MAX_TX_ACCOUNT_LOCKS limit is not exceeded since there are 64 accounts in total.
However, when load_transaction is called, 62 program data accounts will be appended and when the executable accounts are added by load_executable_accounts, the initial vector capacity will be exceeded, so the vector will need to be resized which may result in a degraded cluster performance.

Code Location:

Listing 11: runtime/src/accounts.rs (Lines 358,377)

```
342     if bpf_loader_upgradeable::check_id(account.owner()) {
343         if message.is_writable(i) && !message.
 ↳ is_upgradeable_loader_present() {
344             error_counters.invalid_writable_account += 1;
```

```
345             return Err(TransactionError::InvalidWritableAccount);
346         }
347         if account.executable() {
348             // The upgradeable loader requires the derived
 ↳ ProgramData account
349             if let Ok(UpgradeableLoaderState::Program {
350                 programdata_address,
351             }) = account.state()
352             {
353                 if let Some((programdata_account, _)) = self
354                     .accounts_db
355                     .load_with_fixed_root(ancestors, &
 ↳ programdata_address)
356                 {
357                     account_deps
358                         .push((programdata_address,
 ↳ programdata_account));
359                 } else {
360                     error_counters.account_not_found += 1;
361                     return Err(TransactionError::
 ↳ ProgramAccountNotFound);
362                 }
363             } else {
364                 error_counters.invalid_program_for_execution += 1;
365                 return Err(TransactionError::
 ↳ InvalidProgramForExecution);
366             }
367         }
368     } else if account.executable() && message.is_writable(i) {
369         error_counters.invalid_writable_account += 1;
370         return Err(TransactionError::InvalidWritableAccount);
371     }
372     tx_rent += rent;
373     rent_debits.insert(key, rent, account.lamports());
374     account
375 }
376 };
377 accounts.push((*key, account));
```

Listing 12: runtime/src/accounts.rs (Lines 392,398)

```
392 accounts.append(&mut account_deps);
393 if validated_fee_payer {
394     let program_indices = message
```

```
395         .instructions()
396         .iter()
397         .map(|instruction| {
398             self.load_executable_accounts(
399                 ancestors,
400                 &mut accounts,
401                 instruction.program_id_index as usize,
402                 error_counters,
```

Listing 13: runtime/src/accounts.rs (Line 514)

```
485  fn load_executable_accounts(
486         &self,
487         ancestors: &Ancestors,
488         accounts: &mut Vec<TransactionAccount>,
489         mut program_account_index: usize,
490         error_counters: &mut TransactionErrorMetrics,
491     ) -> Result<Vec<usize>> {
492         let mut account_indices = Vec::new();
493         let mut program_id = match accounts.get(
 ↳ program_account_index) {
494             Some(program_account) => program_account.0,
495             None => {
496                 error_counters.account_not_found += 1;
497                 return Err(TransactionError::
 ↳ ProgramAccountNotFound);
498             }
499         };
500         let mut depth = 0;
501         while !native_loader::check_id(&program_id) {
502             if depth >= 5 {
503                 error_counters.call_chain_too_deep += 1;
504                 return Err(TransactionError::CallChainTooDeep);
505             }
506             depth += 1;
507
508             program_account_index = match self
509                 .accounts_db
510                 .load_with_fixed_root(ancestors, &program_id)
511             {
512                 Some((program_account, _)) => {
513                     let account_index = accounts.len();
514                     accounts.push((program_id, program_account));
515                     account_index
```

```
516                 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to initialize the accounts vector with the maximum theoretically possible capacity so that it does not need to be resized in case of extra allocation.

Remediation Plan:

ACKNOWLEDGED: The Solana team acknowledged this issue.

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 the 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:

- runtime/src/shared_buffer_reader.rs
- runtime/src/snapshot_minimizer.rs
- runtime/src/status_cache.rs
- runtime/src/serde_snapshot.rs
- runtime/src/append_vec.rs
- runtime/src/stake_account.rs
- runtime/src/bank_client.rs
- runtime/src/snapshot_package.rs
- runtime/src/read_only_accounts_cache.rs
- runtime/src/accounts_cache.rs
- runtime/src/accounts_index_storage.rs
- runtime/src/accounts_hash.rs
- runtime/src/bank.rs
- runtime/src/ancient_append_vecs.rs
- runtime/src/snapshot_utils.rs
- runtime/src/bank/sysvar_cache.rs
- runtime/src/hardened_unpack.rs
- runtime/src/accounts_index.rs
- runtime/src/in_mem_accounts_index.rs
- runtime/src/loader_utils.rs

- runtime/src/waitable_condvar.rs
- runtime/src/bucket_map_holder.rs
- runtime/src/cache_hash_data.rs
- runtime/src/accounts_background_service.rs
- runtime/src/serde_snapshot/newer.rs
- runtime/src/serde_snapshot/utils.rs
- runtime/src/secondary_index.rs
- runtime/src/accounts.rs
- runtime/src/bucket_map_holder_stats.rs

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:

ACKNOWLEDGED: The Solana team acknowledged this issue.

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:

ID	package	Short Description
RUSTSEC-2020-0071	time	Potential segfault in the time crate

4.2 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.

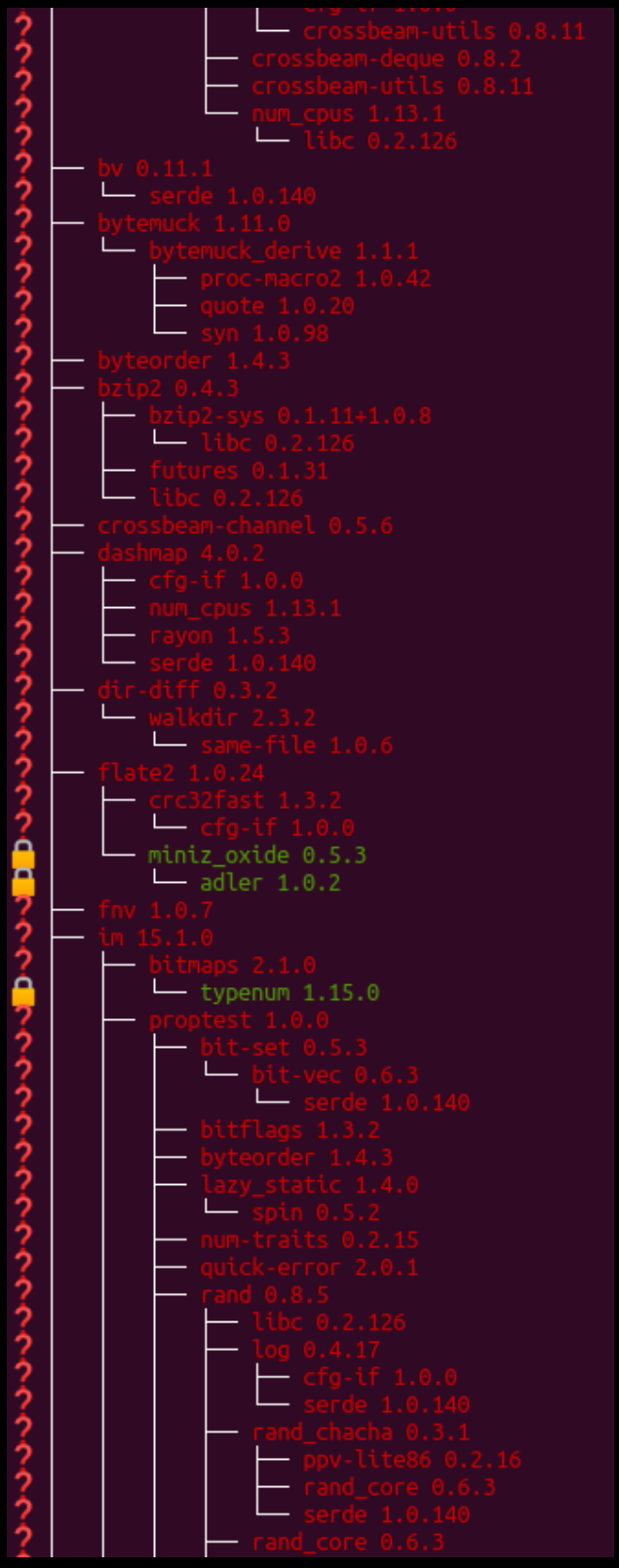
AUTOMATED TESTING

Results:

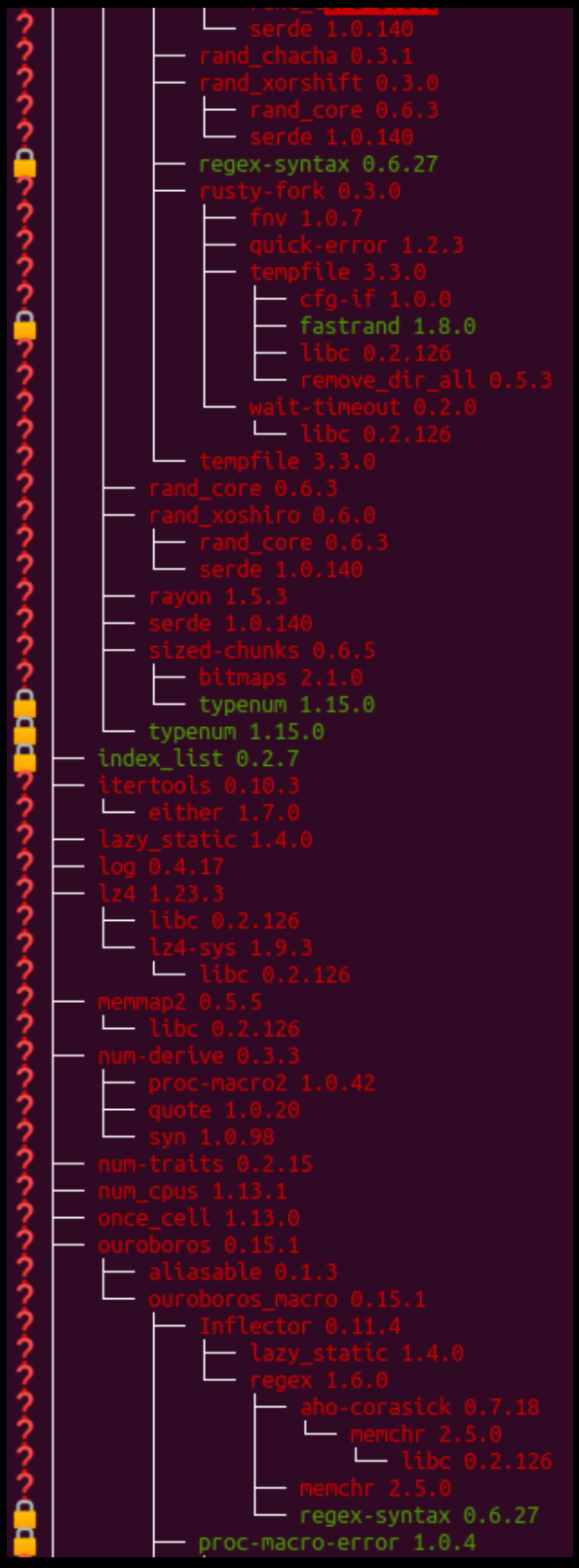
```
Symbols:
    🔒 = All entry point .rs files declare #![forbid(unsafe_code)].
    ❓ = This crate may use unsafe code.

❓ solana-runtime 1.11.3
❓ ├── arrayref 0.3.6
❓ ├── bincode 1.3.3
❓ │   └── serde 1.0.140
❓ │       └── serde_derive 1.0.140
❓ │           ├── proc-macro2 1.0.42
❓ │           │   └── unicode-ident 1.0.2
❓ │           ├── quote 1.0.20
❓ │           │   └── proc-macro2 1.0.42
❓ │           └── syn 1.0.98
❓ │               ├── proc-macro2 1.0.42
❓ │               ├── quote 1.0.20
❓ │               └── unicode-ident 1.0.2
❓ ├── blake3 1.3.1
❓ │   ├── arrayref 0.3.6
❓ │   ├── arrayvec 0.7.2
❓ │   │   └── serde 1.0.140
❓ │   ├── 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 1.0.140
🔒 │   │   │       └── typenum 1.15.0
🔒 │   │   ├── crypto-common 0.1.6
❓ │   │   │   ├── generic-array 0.14.5
❓ │   │   │   └── rand_core 0.6.3
❓ │   │   │       ├── getrandom 0.2.7
❓ │   │   │       │   ├── cfg-if 1.0.0
❓ │   │   │       │   └── libc 0.2.126
❓ │   │   │       └── serde 1.0.140
🔒 │   │   └── typenum 1.15.0
❓ │   └── subtle 2.4.1
❓ └── rayon 1.5.3
❓     ├── crossbeam-deque 0.8.2
❓     │   ├── cfg-if 1.0.0
❓     │   ├── crossbeam-epoch 0.9.10
❓     │   │   ├── cfg-if 1.0.0
❓     │   │   ├── crossbeam-utils 0.8.11
❓     │   │   │   ├── 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 1.0.140
❓     │   │   ├── memoffset 0.6.5
❓     │   │   ├── once_cell 1.13.0
❓     │   │   └── scopeguard 1.1.0
❓     │   └── crossbeam-utils 0.8.11
❓     ├── either 1.7.0
❓     │   └── serde 1.0.140
❓     └── rayon-core 1.9.3
❓         ├── crossbeam-channel 0.5.6
```

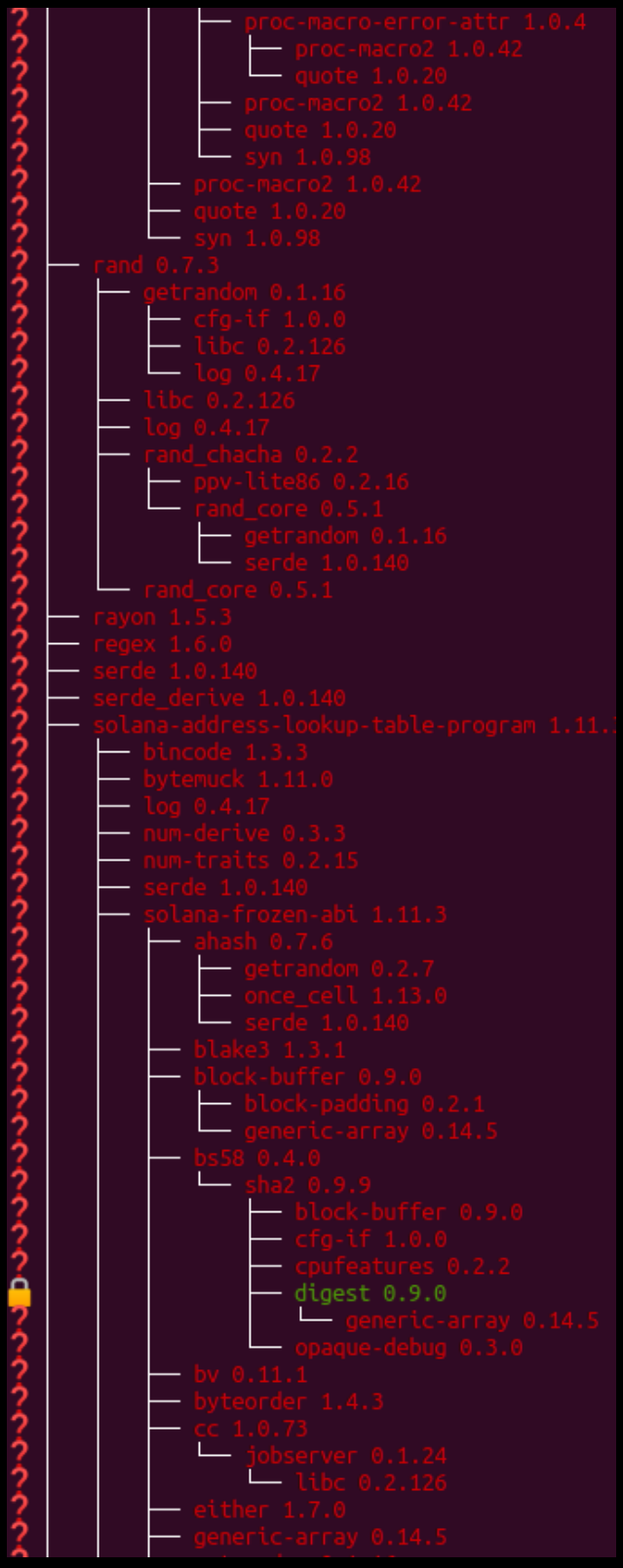
AUTOMATED TESTING



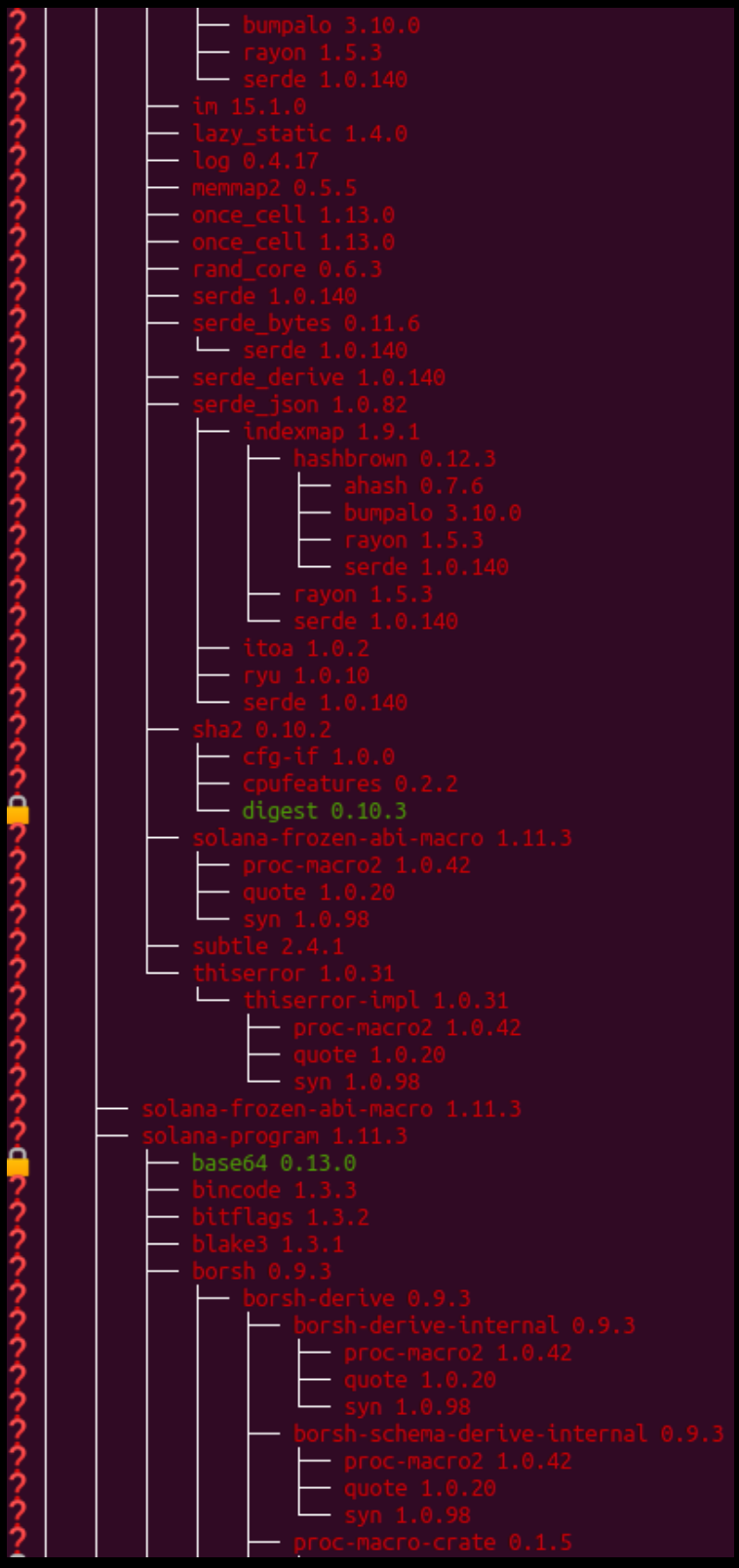
AUTOMATED TESTING



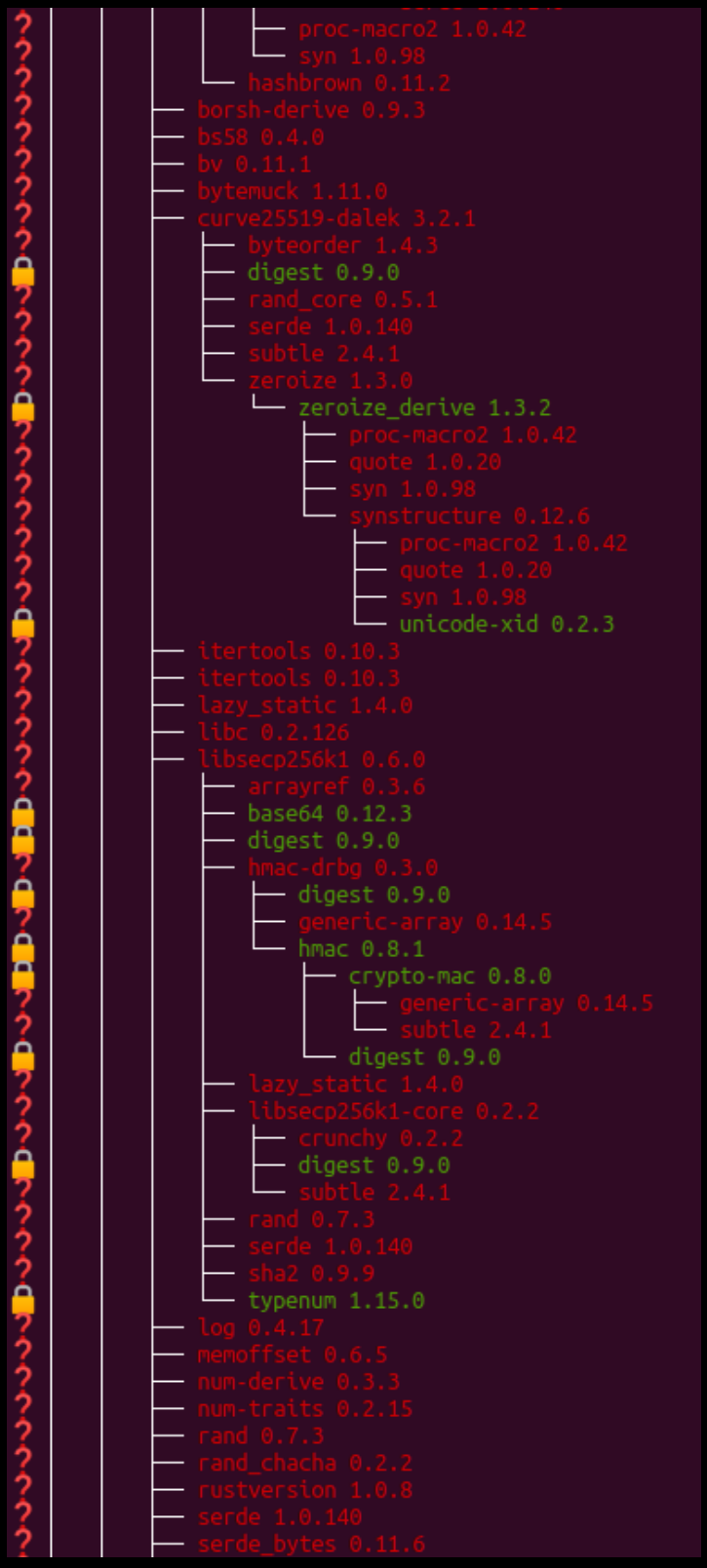
AUTOMATED TESTING



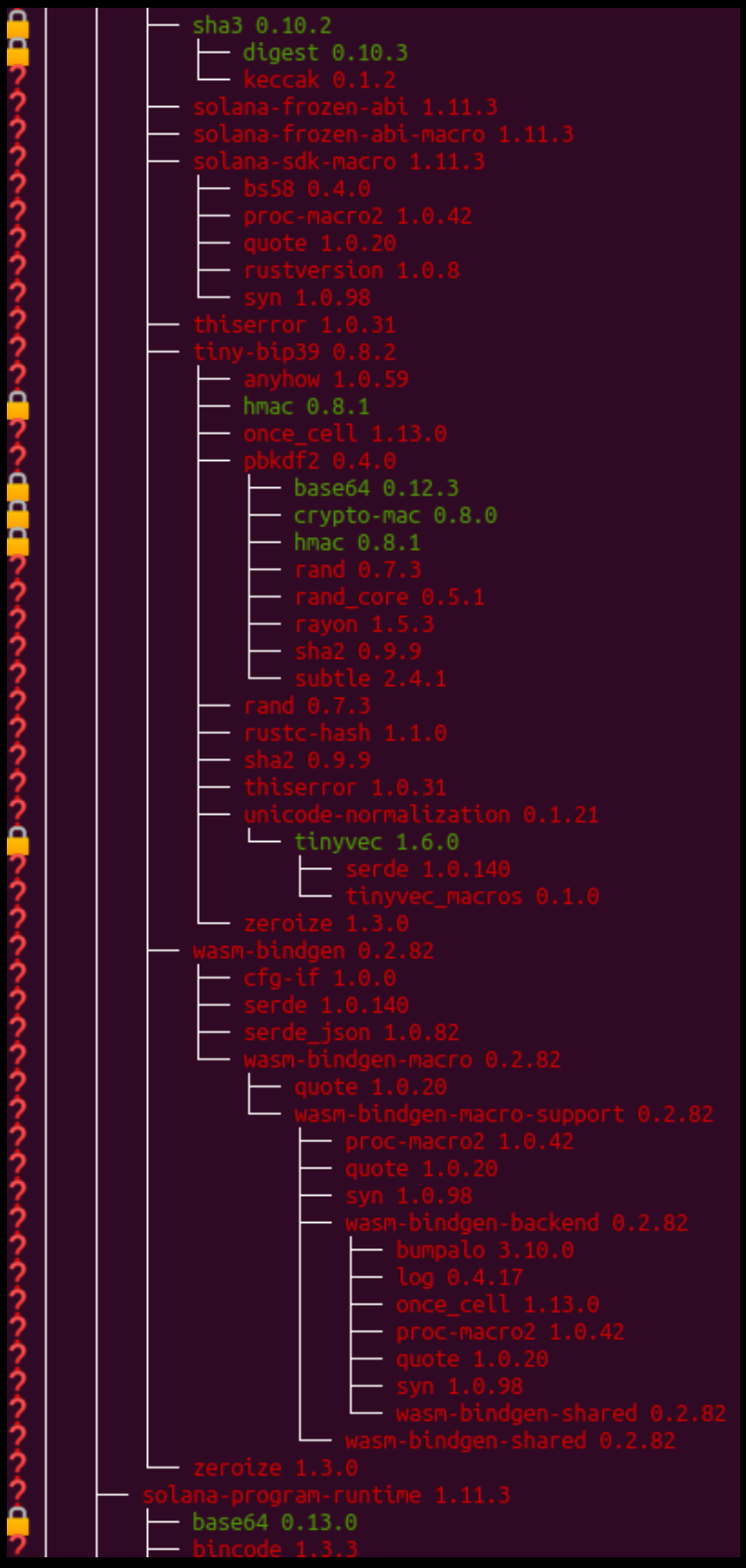
AUTOMATED TESTING



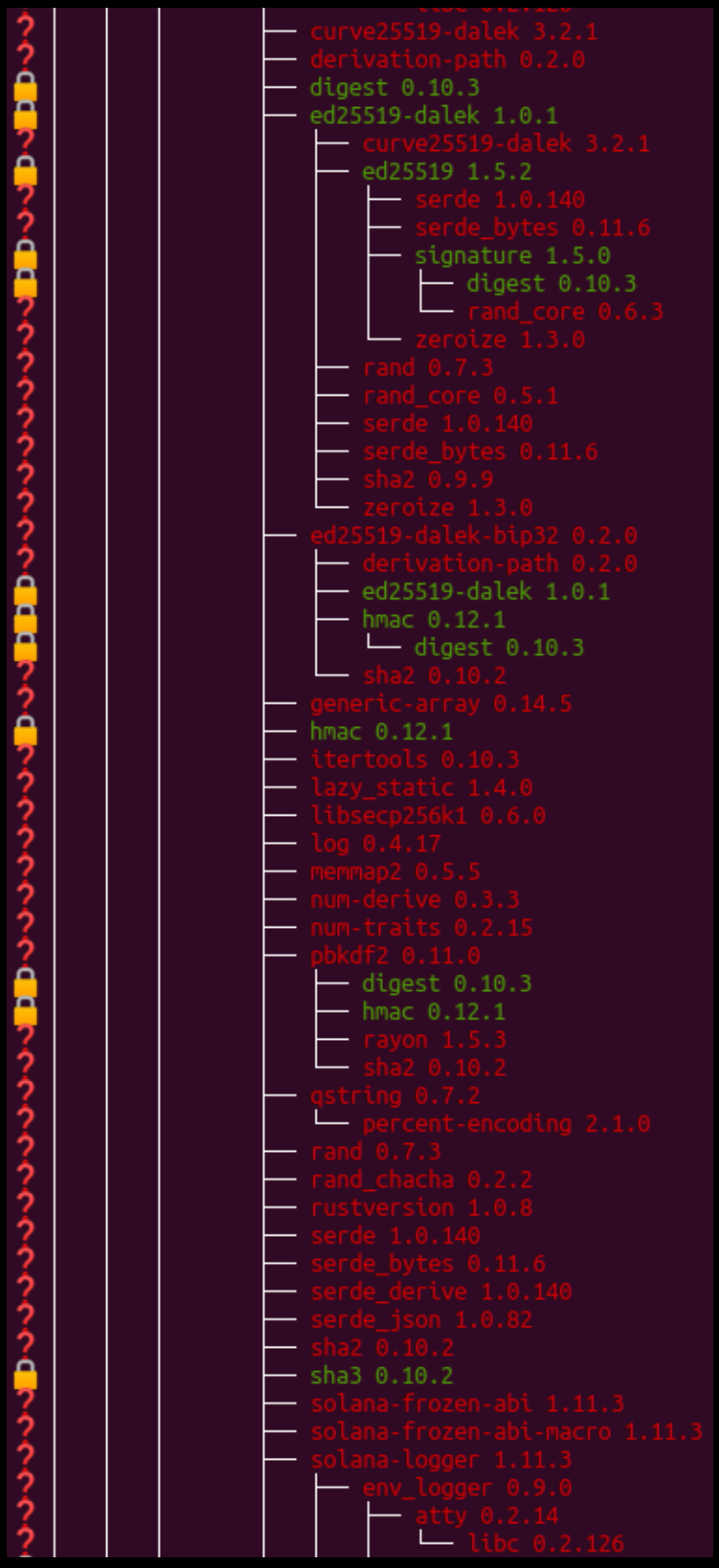
AUTOMATED TESTING



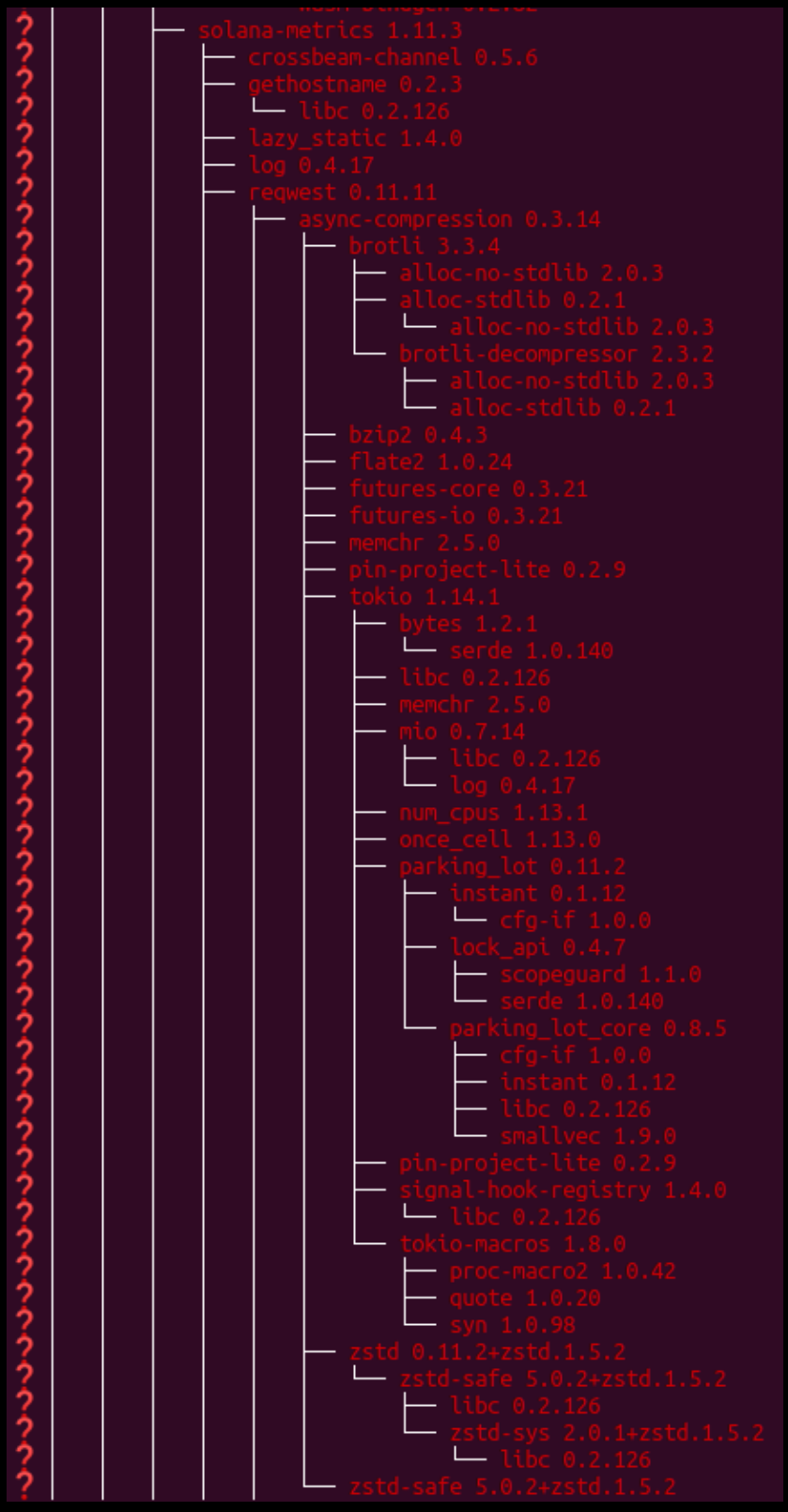
AUTOMATED TESTING



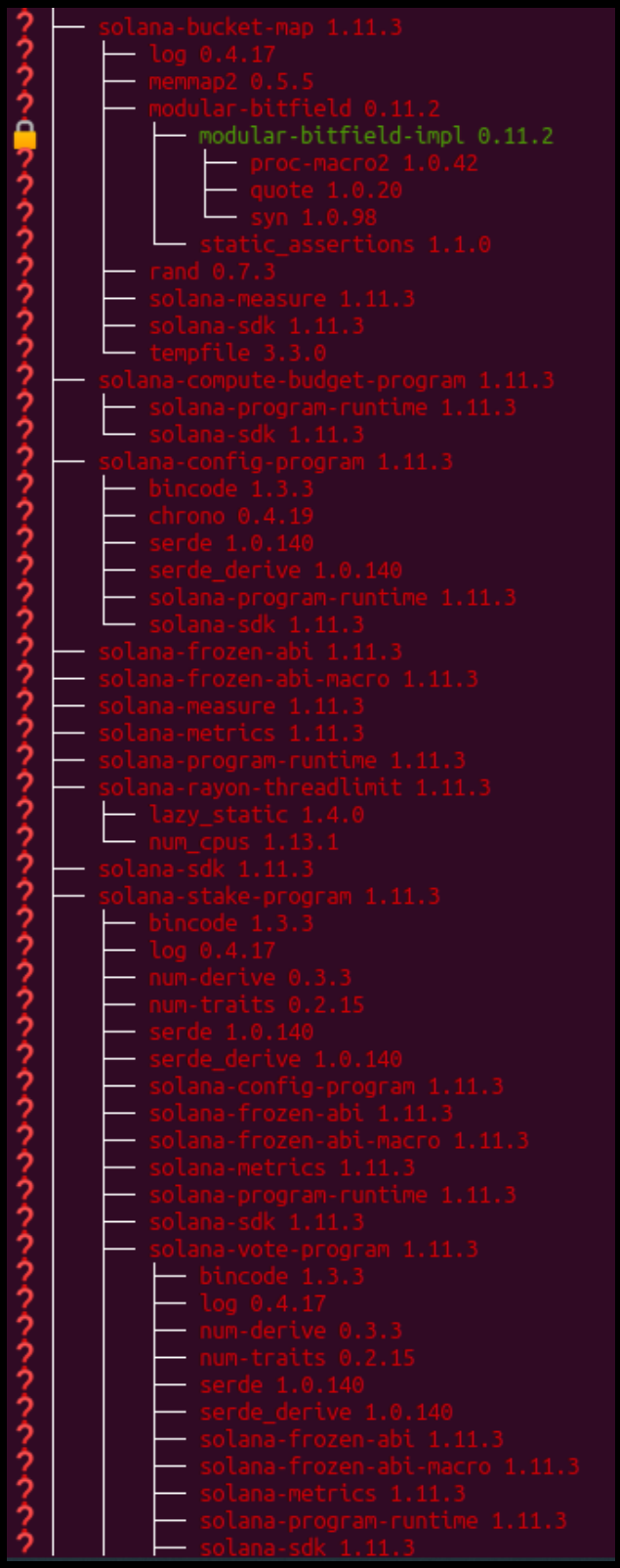
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN