// HALBORN

Solana Labs - Runtime 13107b4 -> 77a56b0

L1 Security Assessment

Prepared by: **Halborn**

Date of Engagement: **May 8th, 2023 - July 5th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/08/2023	Isabel Burruezo
0.2	Document Updates	06/03/2023	Michael Smith
0.3	Final Draft	06/05/2023	Isabel Burruezo
0.4	Draft Review	06/05/2023	Piotr Cielas
0.5	Draft Review	06/05/2023	Gabi Urrutia
1.0	Remediation Plan	08/22/2023	Isabel Burruezo
1.1	Remediation Plan Review	08/22/2023	Piotr Cielas
1.2	Remediation Plan Review	08/23/2023	Gabi Urrutia
1.3	Remediation Plan Updates	08/30/2023	Isabel Burruezo
1.4	Remediation Plan Updates Review	08/30/2023	Piotr Cielas
1.5	Remediation Plan Updates Review	08/31/2023	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
Michael Smith	Halborn	Michael.Smith@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Solana is an open-source project implementing a new, high-performance, permissionless blockchain. Changes in scope affected several modules, the most important ones are briefly described. Sealevel, Solana's parallel smart contracts runtime, is a concurrent transaction processor. Transactions specify their data dependencies upfront, and dynamic memory allocation is explicit. By separating program code from the state it operates on, the runtime can choreograph concurrent access. Gulf Stream the transaction forwarding protocol, which is Solana's mempool-less solution for forwarding and storing transactions before processing them. The Gossip Service acts as a gateway to nodes in the control plane. Validators use the service to ensure information is available to all other nodes in a cluster. TPU (Transaction Processing Unit) is the logic of the validator responsible for block production.

Halborn conducted a security assessment on a set of changes to the Solana repository made between two different commits, beginning on May 8th, 2023 and ending on July 5th, 2023 . The security assessment was scoped to the updates to the master branch of the solana GitHub repository. Commit hashes and further details can be found in the **Scope** section of this report.

1.2 ASSESSMENT SUMMARY

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

The purpose of this assessment is to:

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

In summary, Halborn did not identify any significant issue; however, some recommendations were given to reduce the likelihood and impact of risks, which were acknowledged by Solana Labs .

/In summary, Halborn identified some security risks that were mostly addressed by the Solana Labs team./

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, 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 assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business 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)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Solana L1

- Repository: solana
- Commit IDs:
  - start: 13107b4
  - final: 77a56b0
- Modules in scope:
  1. program-runtime (solana/program-runtime/src)
  2. runtime (solana/runtime/src)
  3. bpf_loader (solana/programs/bpf_loader/src)

Out-of-scope:
- third-party libraries and dependencies
- financial-related attacks

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING CARGO OVERFLOW CHECKS	Informational (0.0)	ACKNOWLEDGED
(HAL-02) COMPUTE UNIT INCONSISTENCY - POTENTIAL OVERFLOW	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL (0.0)

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 on release builds, it will be silenced, leading to unexpected behavior of an application. Even if checked arithmetic is used through checked_* or saturating_*, it is recommended to have that check in Cargo.toml.

Code Location:

- program-runtime/Cargo.toml
- programs/bpf_loader/Cargo.toml

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation:

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

Remediation Plan:

ACKNOWLEDGED: The Solana Labs team acknowledged this issue and have opted not to incur potential performance penalties by enabling overflow checks.

4.2 (HAL-02) COMPUTE UNIT INCONSISTENCY - POTENTIAL OVERFLOW - INFORMATIONAL (0.0)

Description:

The ComputeBudgetInstruction::SetComputeUnitLimit instruction allows users to limit the amount of compute_units a transaction is allowed to use by providing a u32 parameter. This is eventually processed in runtime::bank::Bank::load_and_execute_transactions and program-runtime ::compute_budget::ComputeBudget::process_instructions to calculate the users' Prioritization Fee and return a PrioritizationFeeDetails struct. However, the PrioritizationFeeDetails takes an u64, while the SetComputeUnitLimit takes a u32 and has to be cast as a u64.

Though there is no vulnerability in this scenario, currently the MAX_COMPUTE_UNIT_LIMIT is set to 1400000 which is below u32:MAX, if PrioritizationFeeDetails::priority is changed to the u32 type casting can be avoided. Type casting is a common attack vector that should be avoided if possible, as new vulnerabilities can be accidentally introduced in the future if not handled correctly.

Code Location:

Listing 1: sdk/src/compute_budget.rs (Line 55)

```
54 /// Create a `ComputeBudgetInstruction::SetComputeUnitLimit` `
↳ Instruction`
55 pub fn set_compute_unit_limit(units: u32) -> Instruction {
56     Instruction::new_with_borsh(id(), &Self::SetComputeUnitLimit(
↳ units), vec![])
57 }
```

Listing 2: runtime/src/bank.rs

```
4676 let process_transaction_result = compute_budget.
 ↳ process_instructions(
4677     tx.message().program_instructions_iter(),
4678     true,
4679     !self
4680         .feature_set
4681         .is_active(&remove_deprecated_request_unit_ix::id()),
4682     true, // don't reject txs that use request heap size ix
4683     self.feature_set
4684         .is_active(&
 ↳ add_set_tx_loaded_accounts_data_size_instruction::id()),
4685 );
```

Listing 3: program-runtime/src/compute_budget.rs (Line 218)

```
171 pub fn process_instructions<'a>(
172     &mut self,
173     instructions: impl Iterator<Item = (&'a Pubkey, &'a
 ↳ CompiledInstruction)>,
174     default_units_per_instruction: bool,
175     support_request_units_deprecated: bool,
176     enable_request_heap_frame_ix: bool,
177     support_set_loaded_accounts_data_size_limit_ix: bool,
178 ) -> Result<PrioritizationFeeDetails, TransactionError> {
179     let mut num_non_compute_budget_instructions: usize = 0;
180     let mut updated_compute_unit_limit = None;
181     let mut requested_heap_size = None;
182     let mut prioritization_fee = None;
183     let mut updated_loaded_accounts_data_size_limit = None;
184
185     for (i, (program_id, instruction)) in instructions.enumerate()
 ↳  {
186         if compute_budget::check_id(program_id) {
187             let invalid_instruction_data_error = TransactionError
 ↳ ::InstructionError(
188                 i as u8,
189                 InstructionError::InvalidInstructionData,
190             );
191             let duplicate_instruction_error = TransactionError::
 ↳ DuplicateInstruction(i as u8);
192
193             match try_from_slice_unchecked(&instruction.data) {
```

```
194                 Ok(ComputeBudgetInstruction::
 ↳ RequestUnitsDeprecated {
195                     units: compute_unit_limit,
196                     additional_fee,
197                 }) if support_request_units_deprecated => {
198                     if updated_compute_unit_limit.is_some() {
199                         return Err(duplicate_instruction_error);
200                     }
201                     if prioritization_fee.is_some() {
202                         return Err(duplicate_instruction_error);
203                     }
204                     updated_compute_unit_limit = Some(
 ↳ compute_unit_limit);
205                     prioritization_fee =
206                         Some(PrioritizationFeeType::Deprecated(
 ↳ additional_fee as u64));
207                 }
208                 Ok(ComputeBudgetInstruction::RequestHeapFrame(
 ↳ bytes)) => {
209                     if requested_heap_size.is_some() {
210                         return Err(duplicate_instruction_error);
211                     }
212                     requested_heap_size = Some((bytes, i as u8));
213                 }
214                 Ok(ComputeBudgetInstruction::SetComputeUnitLimit(
 ↳ compute_unit_limit)) => {
215                     if updated_compute_unit_limit.is_some() {
216                         return Err(duplicate_instruction_error);
217                     }
218                     updated_compute_unit_limit = Some(
 ↳ compute_unit_limit);
219                 }
220                 Ok(ComputeBudgetInstruction::SetComputeUnitPrice(
 ↳ micro_lamports)) => {
221                     if prioritization_fee.is_some() {
222                         return Err(duplicate_instruction_error);
223                     }
224                     prioritization_fee =
225                         Some(PrioritizationFeeType::
 ↳ ComputeUnitPrice(micro_lamports));
226                 }
227                 Ok(ComputeBudgetInstruction::
 ↳ SetLoadedAccountsDataSizeLimit(bytes))
228                     if
```

```
 ↳ support_set_loaded_accounts_data_size_limit_ix =>
229                 {
230                     if updated_loaded_accounts_data_size_limit.
 ↳ is_some() {
231                         return Err(duplicate_instruction_error);
232                     }
233                     updated_loaded_accounts_data_size_limit = Some
 ↳ (bytes as usize);
234                 }
235                 _ => return Err(invalid_instruction_data_error),
236             }
237         } else {
238             // only include non-request instructions in default
 ↳ max calc
239             num_non_compute_budget_instructions =
240                 num_non_compute_budget_instructions.saturating_add
 ↳ (1);
241         }
242     }
243
244     if let Some((bytes, i)) = requested_heap_size {
245         if !enable_request_heap_frame_ix
246             || bytes > MAX_HEAP_FRAME_BYTES
247             || bytes < MIN_HEAP_FRAME_BYTES as u32
248             || bytes % 1024 != 0
249         {
250             return Err(TransactionError::InstructionError(
251                 i,
252                 InstructionError::InvalidInstructionData,
253             ));
254         }
255         self.heap_size = Some(bytes as usize);
256     }
257
258     self.compute_unit_limit = if default_units_per_instruction {
259         updated_compute_unit_limit.or_else(|| {
260             Some(
261                 (num_non_compute_budget_instructions as u32)
262                     .saturating_mul(
 ↳ DEFAULT_INSTRUCTION_COMPUTE_UNIT_LIMIT),
263             )
264         })
265     } else {
266         updated_compute_unit_limit
```

```
267     }
268     .unwrap_or(MAX_COMPUTE_UNIT_LIMIT)
269     .min(MAX_COMPUTE_UNIT_LIMIT) as u64;
270
271     self.loaded_accounts_data_size_limit =
 ↳ updated_loaded_accounts_data_size_limit
272         .unwrap_or(MAX_LOADED_ACCOUNTS_DATA_SIZE_BYTES)
273         .min(MAX_LOADED_ACCOUNTS_DATA_SIZE_BYTES);
274
275     Ok(prioritization_fee
276         .map(|fee_type| PrioritizationFeeDetails::new(fee_type,
 ↳ self.compute_unit_limit))
277         .unwrap_or_default())
278 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.0)

Recommendation:

It is recommended to avoid type casting between PrioritizationFeeDetails and user provided data by using the same type for compute units/priority.

Remediation Plan:

ACKNOWLEDGED: The Solana Labs team acknowledged this issue.

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.

5.1 ACCOUNT DATA DIRECT MAPPING IN THE BPF LOADER

Description:

In commit 117a194 a new feature was added to allow direct mapping of account data into vm memory at vm setup time. This feature improves performance as it removes the need to have large copies during transaction execution.

Several tests were performed to ensure that account data is correctly mapped in the memory and calculations are correct, so there are no problems with empty parts (holes) in the memory that for. example could be caused by changing account size (in ex. account shrinking).

Results:

No code vulnerabilities were identified.

```
running 1 test
Serialize parameters with memory mapping
Deserialize unaligned
Verify programID after deserialization
OK
Verify instruction data after deserialization
OK
Verify account parameters after deserialization
Account 1, OK
Account 2, OK
Account 3, OK
Account 4, OK
Account 5, OK
Account 6, OK
Account 7, OK
Account 8, OK
test serialization::tests::security_memory_alignment_unaligned ... ok
```

MANUAL TESTING

```
running 1 test
Account serialization with memory mapping
Deserialize mapped regions in memory (aligned)
Verify program id after deserialization
OK
Verify instruction data after deserialization
OK
Verify BPF offset
OK
Verify each account
Account 1, OK
Account 2, OK
Account 3, OK
Account 4, OK
Account 5, OK
Account 6, OK
Account 7, OK
Account 8, OK
test serialization::tests::security_memory_alignment ... ok
```

```
warning: unused variable: `tx`
  --> src/main.rs:88:13
   |
88 |     let mut tx = (&mut rpc_client).transaction_from_instructions(&[ix], &deployer, vec![&deployer])
   |             ^^ help: if this is intentional, prefix it with an underscore: `_tx`

warning: `security-tests` (bin "security-tests") generated 13 warnings
    Finished dev [unoptimized + debuginfo] target(s) in 2.31s
     Running `target/debug/security-tests`
Deploying base program
Sanitize check - counter instruction
OK
Deploying upgrade - program account shrink
Sanitize check - counter instruction
OK
Redeploying base program - account extend / relock
Sanitize check - counter instruction
OK
```

5.2 OVER/UNDER FLOW COMPUTE UNITS & PRIORITY FEES

In commit 696d770 it was observed that there was inconsistency between ComputeBudgetInstruction::SetComputeUnitLimit instruction and PrioritizationFeeDetails::priority types which required type casting when setting the compute_limit for users transactions.

Due to this, we investigated every instance of PrioritizationFeeType to see if there were typecasting vulnerabilities that would result in an under/overflow, but were unable to find any. Further tests were done to manipulate the fees in unexpected ways.

Results:

No code vulnerabilities were identified

```
running 1 test
User submitted compute_unit_limit = Some(4294967295)
Final compute_unit_limit: 1400000

prioritization_fee 2800000000 +
write_lock_fee 0 +
compute_fee 0 +
* congestion_multiplier 1
total fees = 2800000001

thread 'bank::tests::test_calculate_prioritization_fee_overflow' panicked at 'attempt to add with overflow', runtime/src/bank/tests.rs:10706:9
stack backtrace:
   0: rust_begin_unwind
             at /rustc/84c898d65adf2f39a5a98507f1fe0ce10a2b8dbc/library/std/src/panicking.rs:579:5
   1: core::panicking::panic_fmt
             at /rustc/84c898d65adf2f39a5a98507f1fe0ce10a2b8dbc/library/core/src/panicking.rs:64:14
   2: core::panicking::panic
             at /rustc/84c898d65adf2f39a5a98507f1fe0ce10a2b8dbc/library/core/src/panicking.rs:114:5
   3: solana_runtime::bank::tests::test_calculate_prioritization_fee_overflow
             at ./src/bank/tests.rs:10706:9
   4: solana_runtime::bank::tests::test_calculate_prioritization_fee_overflow::{{closure}}
             at ./src/bank/tests.rs:10670:49
   5: core::ops::function::FnOnce::call_once
             at /rustc/84c898d65adf2f39a5a98507f1fe0ce10a2b8dbc/library/core/src/ops/function.rs:250:5
   6: core::ops::function::FnOnce::call_once
             at /rustc/84c898d65adf2f39a5a98507f1fe0ce10a2b8dbc/library/core/src/ops/function.rs:250:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
test bank::tests::test_calculate_prioritization_fee_overflow - should panic ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 871 filtered out; finished in 0.12s
```

5.3 VALIDATE FEE PAYER

In commit b4331ae the checked_arithmetic_in_fee_validation feature gate was added to validate_fee_payer as well as the use of the arithmetic check use if it is enabled. The purpose of this is to correctly and safely calculate the amount of the payer's lamports when paying the corresponding fees.
This change was reviewed and tested to ensure that no security risks have been accidentally introduced beside the proper and expected calculation's result.

Results:

No code vulnerabilities were identified

```
running 1 test
[+] Validating Fee Payer ...
Payer nonce?: true
Payer Init Balance: 832
Feature Checked Arithmmetic Enable?: true
Fee: 416
Payer Init Balance: 832
payer_account lamports: 832
payer Account lamports post rent: 416
[+] Validating Fee Payer ...
Payer nonce?: true
Payer Init Balance: 832
Feature Checked Arithmmetic Enable?: false
Fee: 416
Payer Init Balance: 832
payer_account lamports: 832
payer Account lamports post rent: 416
[+] Validating Fee Payer ...
Payer nonce?: false
Payer Init Balance: 832
Feature Checked Arithmmetic Enable?: false
Fee: 416
Payer Init Balance: 832
payer_account lamports: 832
payer Account lamports post rent: 416
[+] Validating Fee Payer ...
Payer nonce?: false
Payer Init Balance: 832
Feature Checked Arithmmetic Enable?: false
Fee: 416
Payer Init Balance: 832
payer_account lamports: 832
payer Account lamports post rent: 416
test accounts::tests::test_validate_fee_payer_All_OK ... ok
```

MANUAL TESTING

```
running 1 test
[+] Validating Fee Payer ...
Payer nonce?: true
Payer Init Balance: 416
Feature Checked Arithmmetic Enable?: true
Fee: 0
Payer Init Balance: 416
payer_account lamports: 416
payer Account lamports post rent: 416
[+] Validating Fee Payer ...
Payer nonce?: true
Payer Init Balance: 416
Feature Checked Arithmmetic Enable?: false
Fee: 0
Payer Init Balance: 416
payer_account lamports: 416
payer Account lamports post rent: 416
[+] Validating Fee Payer ...
Payer nonce?: false
Payer Init Balance: 416
Feature Checked Arithmmetic Enable?: false
Fee: 0
Payer Init Balance: 416
payer_account lamports: 416
payer Account lamports post rent: 416
[+] Validating Fee Payer ...
Payer nonce?: false
Payer Init Balance: 416
Feature Checked Arithmmetic Enable?: false
Fee: 0
Payer Init Balance: 416
payer_account lamports: 416
payer Account lamports post rent: 416
test accounts::tests::test_validate_fee_payer_zero_fee ... ok
```

5.4 OWNERSHIP CHAINS

In commit 86fb2d7 a new restriction was added to the feature gate disable_builtin_loader_ownership_chains to ensure the owner of the program is owned itself by the native-loader.

Tests were done to ensure the restriction works as described and cannot be bypassed, for example by chaining with different accounts and loaders.

Results:

No code vulnerabilities were identified

```
running 1 test
disable_builtin_loader_ownership_chains true

Scenario #1: !native_loader::check_id(owner_account.owner())
loaded_accounts: 1
Expected 1 loaded_accounts error found: 1
disable_builtin_loader_ownership_chains true

Scenario #2: !owner_account.executable()
loaded_accounts: 1
Expected 1 loaded_accounts error found: 1
disable_builtin_loader_ownership_chains true

Scenario #3: BPF Loaders
loaded_accounts: 1
Expected 0 loaded_accounts error found: 0
disable_builtin_loader_ownership_chains true

Scenario #4: Bad loader
loaded_accounts: 1
Expected 1 account_not_found error found: 1
test accounts::tests::test_disable_builtin_loader_ownership_chains ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 871 filtered out; finished in 0.05s
```

AUTOMATED TESTING

6.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with the 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-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2023-0001	tokio	Configuration corruption

6.2 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.

Results:

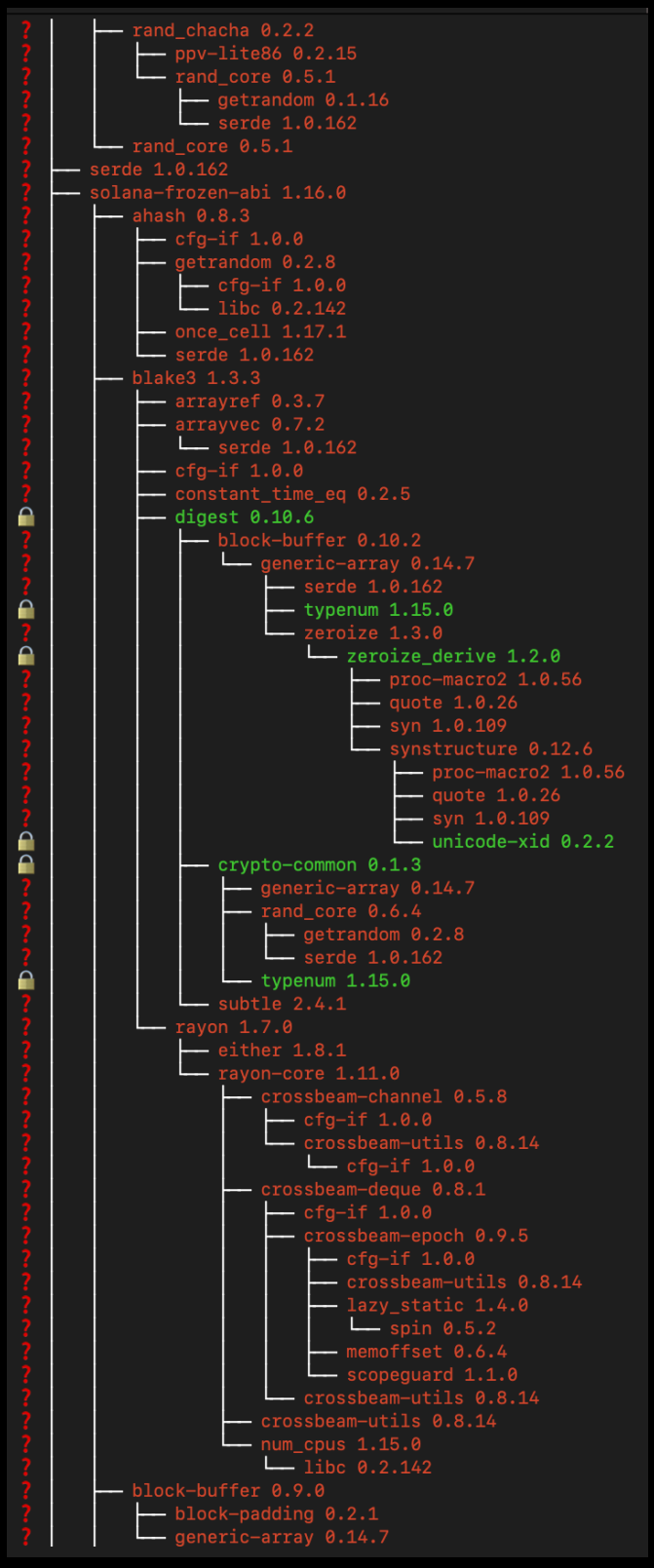
program-runtime

AUTOMATED TESTING

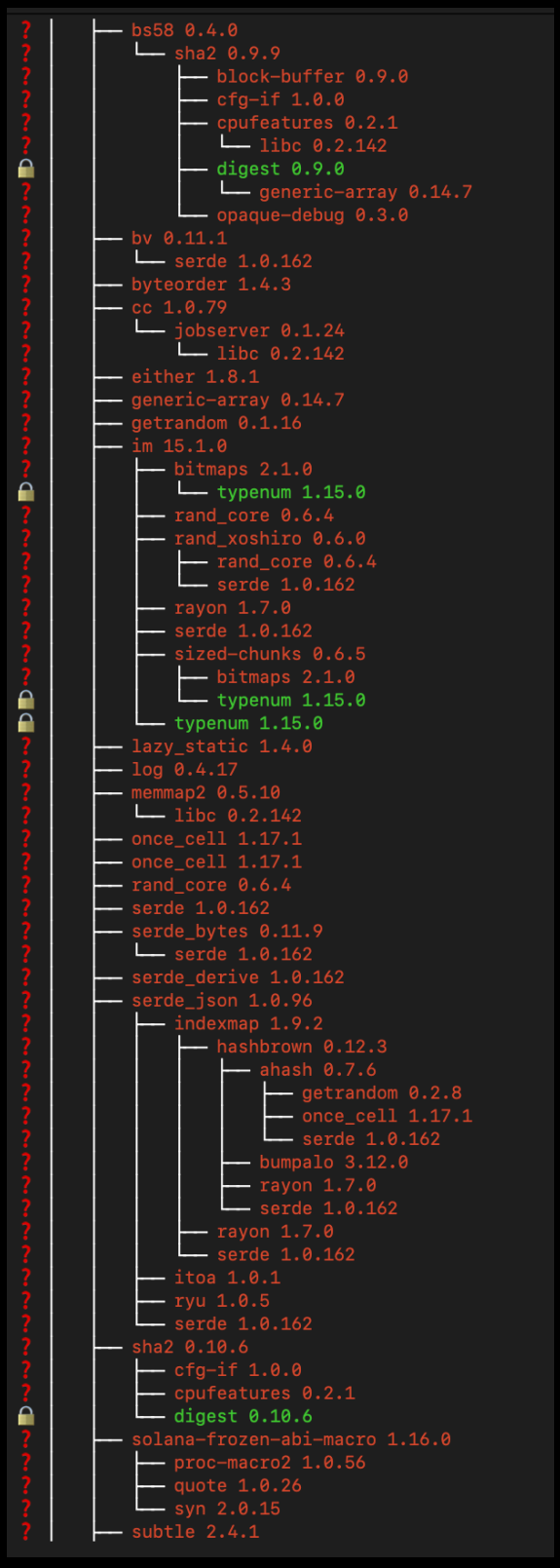
```
Symbols:
    🔒  = All entry point .rs files declare #![forbid(unsafe_code)].
    ❓  = This crate may use unsafe code.

❓ solana-program-runtime 1.16.0
🔒 ├── base64 0.13.1
❓ ├── bincode 1.3.3
❓ │   └── serde 1.0.162
❓ │       └── serde_derive 1.0.162
❓ │           ├── proc-macro2 1.0.56
❓ │           │   └── unicode-ident 1.0.2
❓ │           ├── quote 1.0.26
❓ │           │   └── proc-macro2 1.0.56
❓ │           └── syn 2.0.15
❓ │               ├── proc-macro2 1.0.56
❓ │               ├── quote 1.0.26
❓ │               └── unicode-ident 1.0.2
❓ ├── eager 0.1.0
❓ ├── enum-iterator 1.4.0
❓ │   └── enum-iterator-derive 1.2.0
❓ │       ├── proc-macro2 1.0.56
❓ │       ├── quote 1.0.26
❓ │       └── syn 1.0.109
❓ │           ├── proc-macro2 1.0.56
❓ │           ├── quote 1.0.26
❓ │           └── unicode-ident 1.0.2
❓ ├── itertools 0.10.5
❓ │   └── either 1.8.1
❓ │       └── serde 1.0.162
❓ ├── libc 0.2.142
❓ ├── log 0.4.17
❓ │   ├── cfg-if 1.0.0
❓ │   └── serde 1.0.162
❓ ├── num-derive 0.3.3
❓ │   ├── proc-macro2 1.0.56
❓ │   ├── quote 1.0.26
❓ │   └── syn 1.0.109
❓ ├── num-traits 0.2.15
❓ │   └── libm 0.2.1
❓ ├── percentage 0.1.0
❓ │   └── num 0.2.1
❓ │       ├── num-bigint 0.2.6
❓ │       │   ├── num-integer 0.1.44
❓ │       │   │   └── num-traits 0.2.15
❓ │       │   ├── num-traits 0.2.15
❓ │       │   └── serde 1.0.162
❓ │       ├── num-complex 0.2.4
❓ │       │   ├── num-traits 0.2.15
❓ │       │   └── serde 1.0.162
❓ │       ├── num-integer 0.1.44
❓ │       ├── num-iter 0.1.43
❓ │       │   ├── num-integer 0.1.44
❓ │       │   └── num-traits 0.2.15
❓ │       ├── num-rational 0.2.4
❓ │       │   ├── num-bigint 0.2.6
❓ │       │   ├── num-integer 0.1.44
❓ │       │   ├── num-traits 0.2.15
❓ │       │   └── serde 1.0.162
❓ │       └── num-traits 0.2.15
❓ ├── rand 0.7.3
❓ │   ├── getrandom 0.1.16
❓ │   │   ├── cfg-if 1.0.0
❓ │   │   ├── libc 0.2.142
❓ │   │   └── log 0.4.17
```

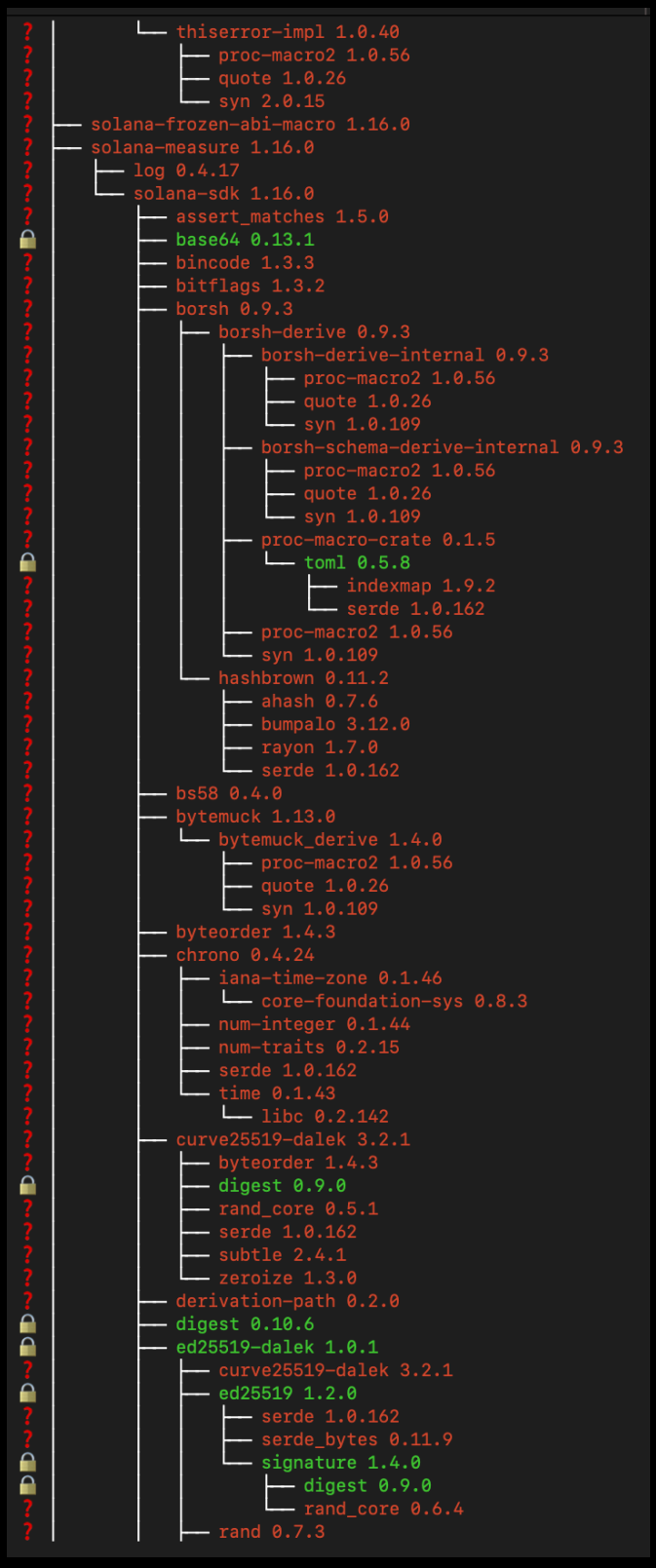
AUTOMATED TESTING



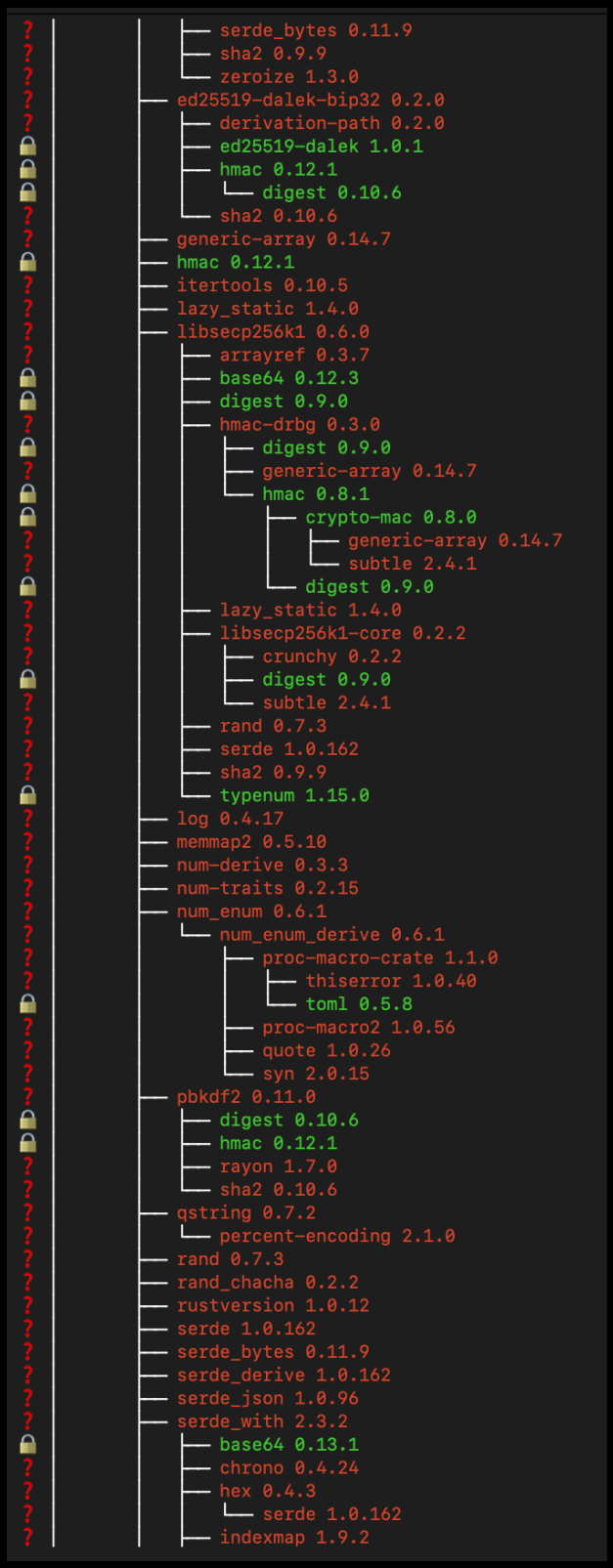
AUTOMATED TESTING



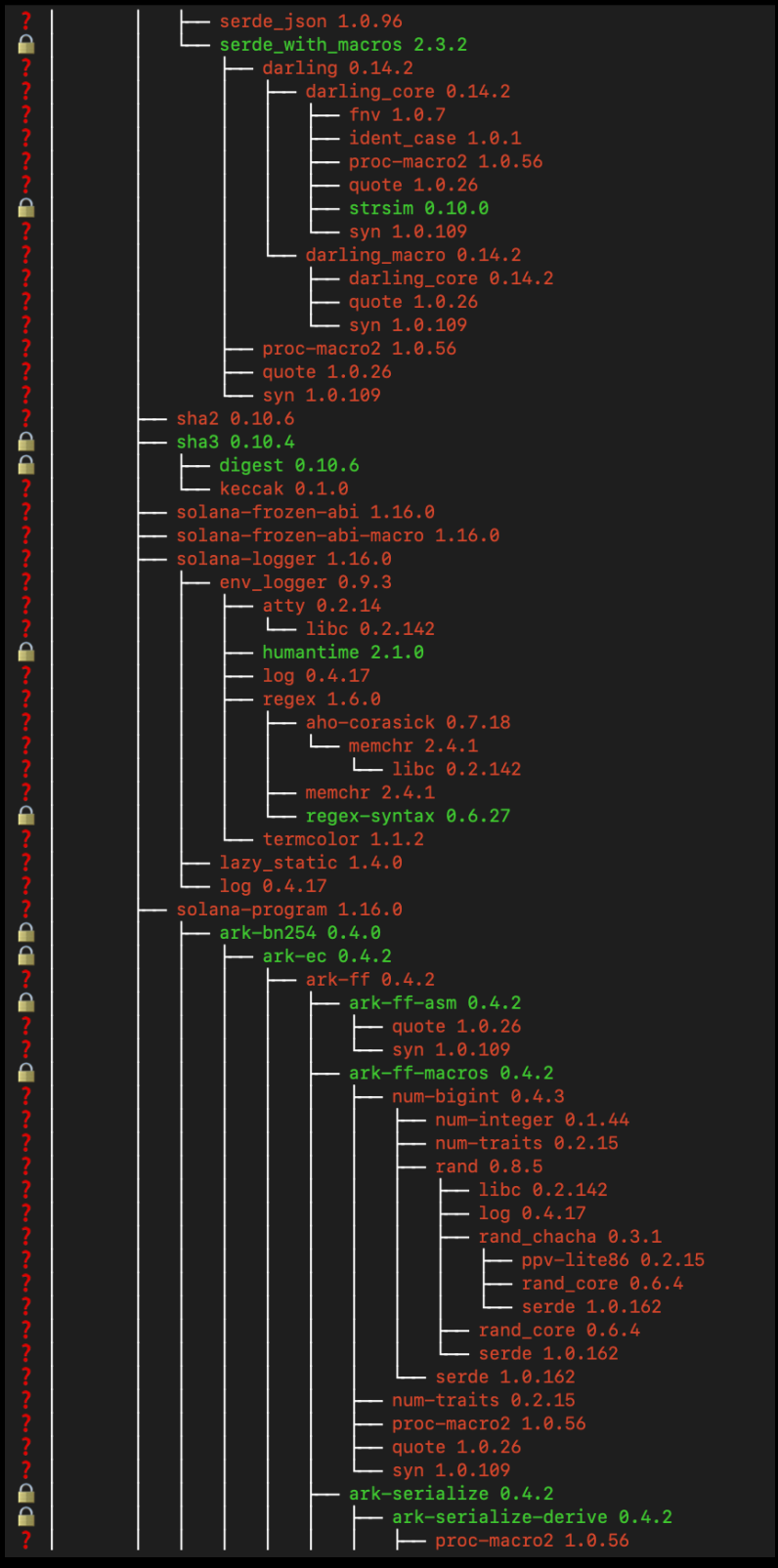
AUTOMATED TESTING



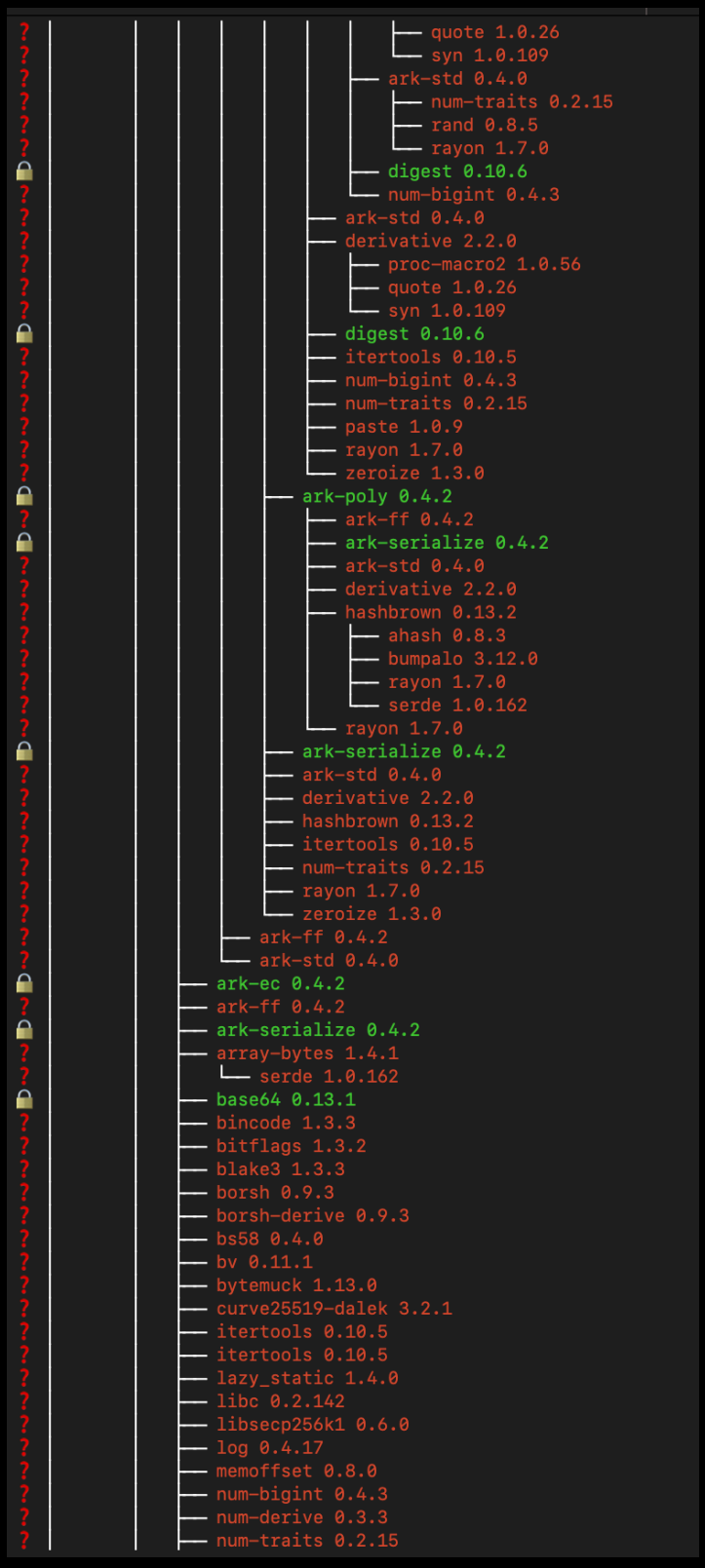
AUTOMATED TESTING



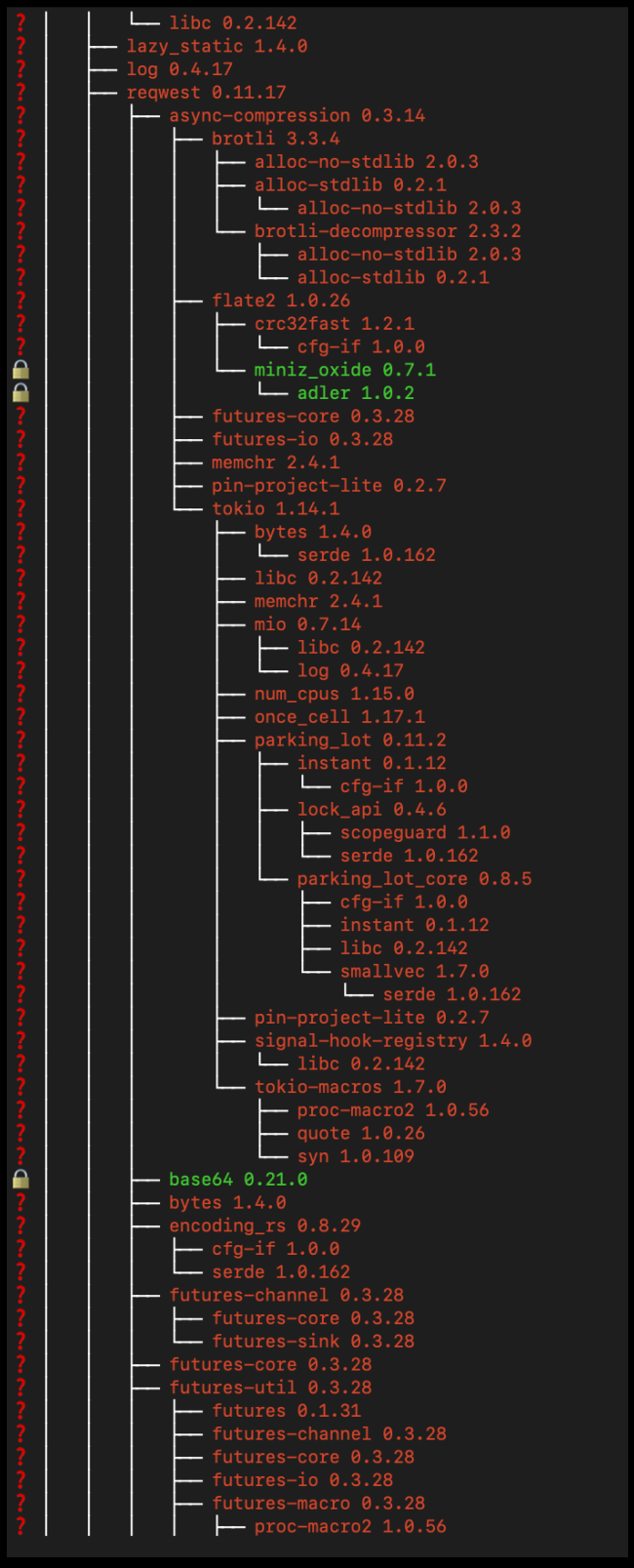
AUTOMATED TESTING



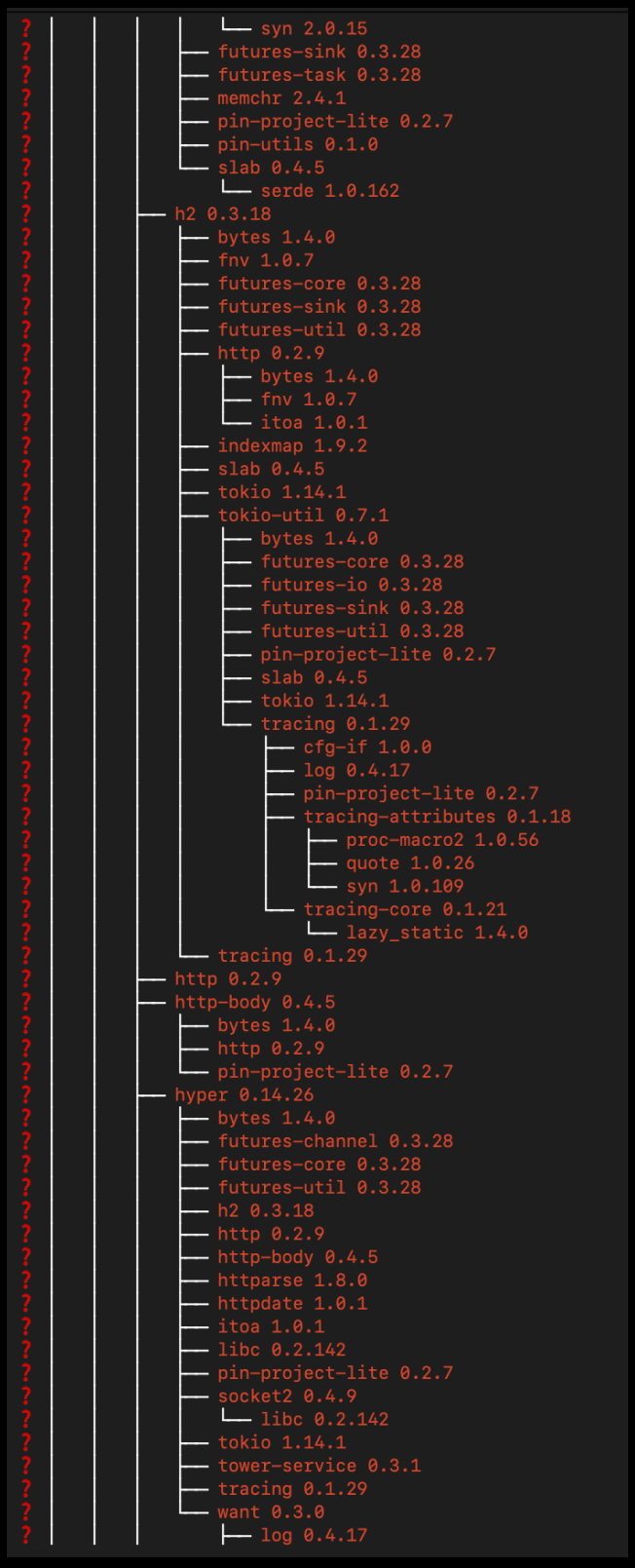
AUTOMATED TESTING



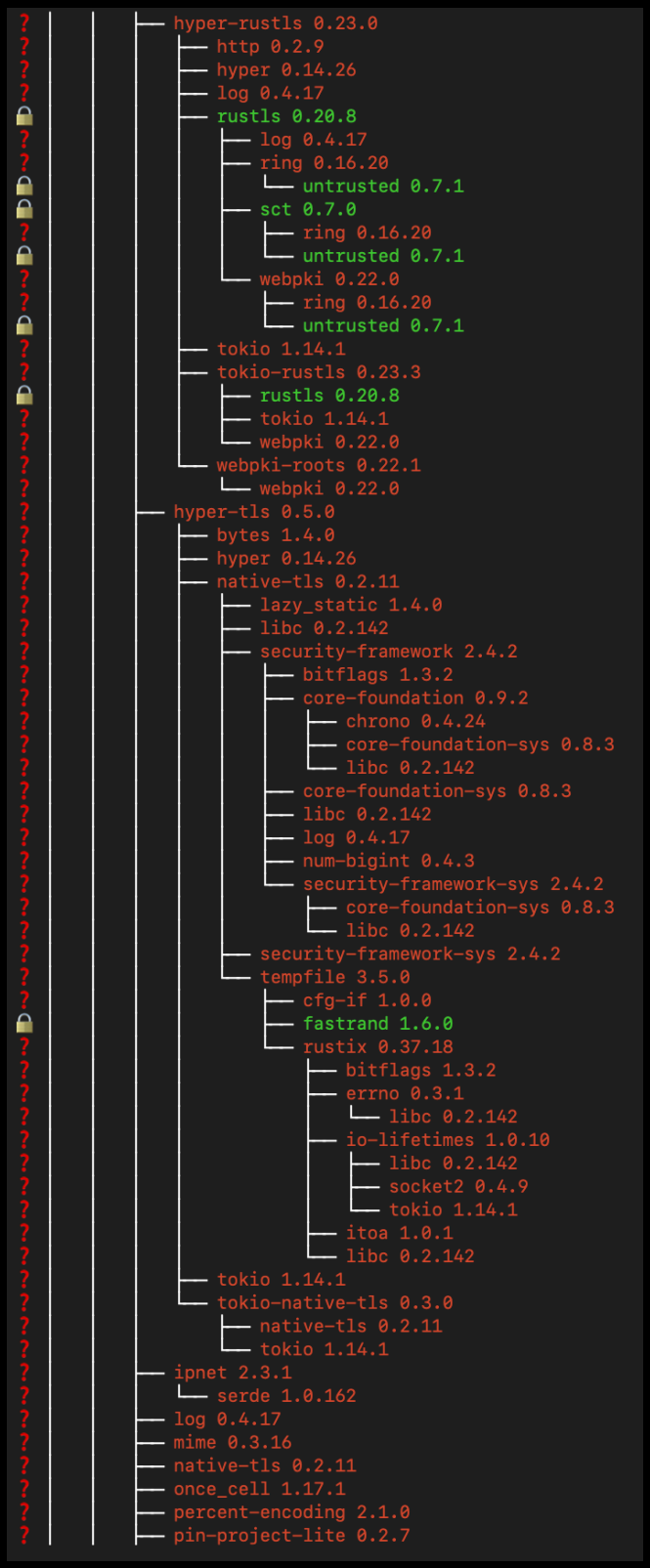
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING

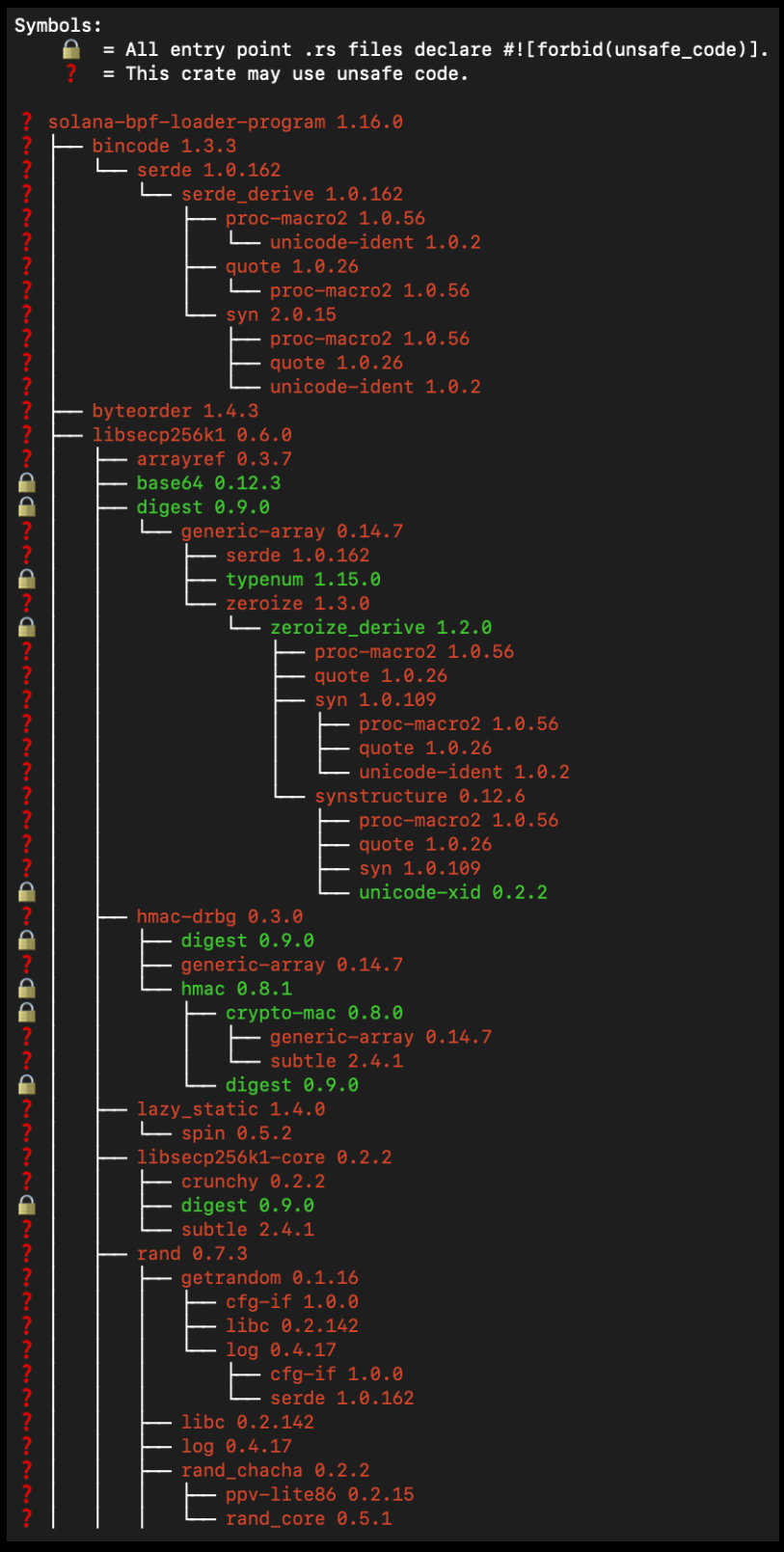


AUTOMATED TESTING

```
🔓 │   │   ├── rustls 0.20.8
🔓 │   │   ├── rustls-pemfile 1.0.0
🔓 │   │   │   └── base64 0.13.1
❓ │   │   ├── serde 1.0.162
❓ │   │   ├── serde_json 1.0.96
🔓 │   │   ├── serde_urlencoded 0.7.1
❓ │   │   │   ├── form_urlencoded 1.0.1
❓ │   │   │   │   ├── matches 0.1.10
❓ │   │   │   │   └── percent-encoding 2.1.0
❓ │   │   │   ├── itoa 1.0.1
❓ │   │   │   ├── ryu 1.0.5
❓ │   │   │   └── serde 1.0.162
❓ │   │   ├── tokio 1.14.1
❓ │   │   ├── tokio-native-tls 0.3.0
❓ │   │   ├── tokio-rustls 0.23.3
❓ │   │   ├── tokio-util 0.7.1
❓ │   │   ├── tower-service 0.3.1
❓ │   │   ├── url 2.2.2
❓ │   │   │   ├── form_urlencoded 1.0.1
❓ │   │   │   ├── idna 0.2.3
❓ │   │   │   │   ├── matches 0.1.10
🔓 │   │   │   │   ├── unicode-bidi 0.3.7
❓ │   │   │   │   │   └── serde 1.0.162
❓ │   │   │   │   └── unicode-normalization 0.1.19
❓ │   │   │   ├── matches 0.1.10
❓ │   │   │   ├── percent-encoding 2.1.0
❓ │   │   │   └── serde 1.0.162
❓ │   │   └── webpki-roots 0.22.1
❓ │   └── solana-sdk 1.16.0
❓ ├── solana-sdk 1.16.0
❓ ├── solana_rbpf 0.3.0
❓ │   ├── byteorder 1.4.3
❓ │   ├── combine 3.8.1
❓ │   │   ├── ascii 0.9.3
❓ │   │   │   └── serde 1.0.162
❓ │   │   ├── byteorder 1.4.3
❓ │   │   ├── either 1.8.1
❓ │   │   ├── memchr 2.4.1
❓ │   │   └── unreachable 1.0.0
❓ │   │       └── void 1.0.2
❓ │   ├── gdbstub 0.6.3
❓ │   │   ├── bitflags 1.3.2
❓ │   │   ├── cfg-if 1.0.0
❓ │   │   ├── log 0.4.17
❓ │   │   ├── managed 0.8.0
❓ │   │   ├── num-traits 0.2.15
❓ │   │   └── paste 1.0.9
❓ │   ├── goblin 0.5.1
❓ │   │   ├── log 0.4.17
❓ │   │   ├── plain 0.2.3
❓ │   │   └── scroll 0.11.0
❓ │   │       └── scroll_derive 0.11.0
❓ │   │           ├── proc-macro2 1.0.56
❓ │   │           ├── quote 1.0.26
❓ │   │           └── syn 1.0.109
❓ │   ├── hash32 0.2.1
❓ │   │   └── byteorder 1.4.3
❓ │   ├── libc 0.2.142
❓ │   ├── log 0.4.17
❓ │   ├── rand 0.8.5
❓ │   ├── rustc-demangle 0.1.21
❓ │   ├── scroll 0.11.0
❓ │   ├── thiserror 1.0.40
❓ │   └── winapi 0.3.9
❓ └── thiserror 1.0.40
```

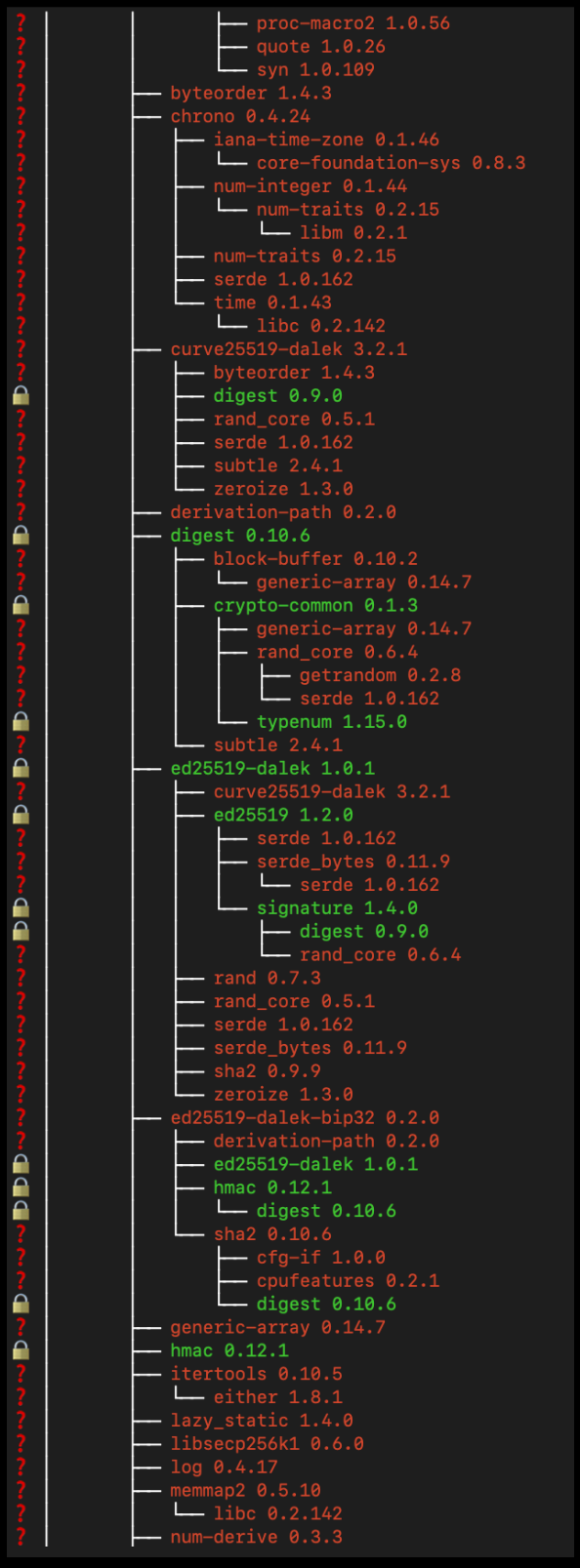
bpf_loader

AUTOMATED TESTING

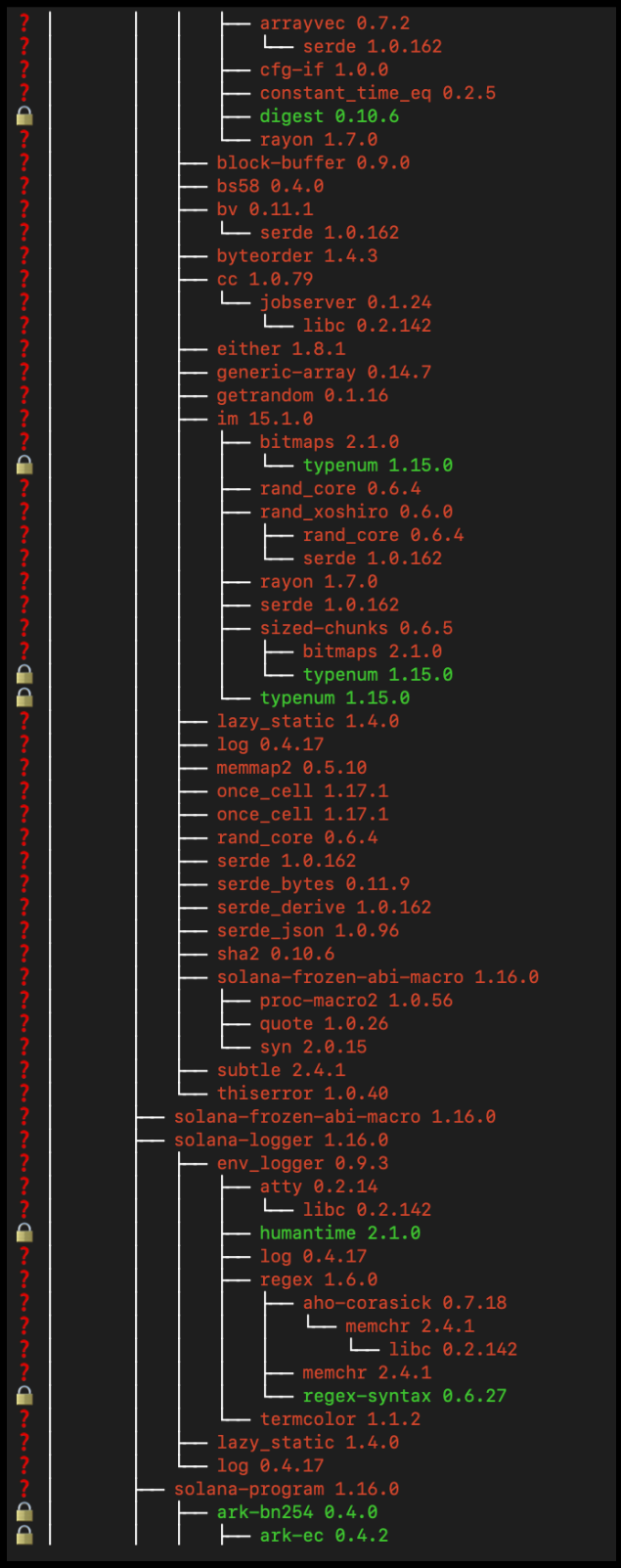


AUTOMATED TESTING

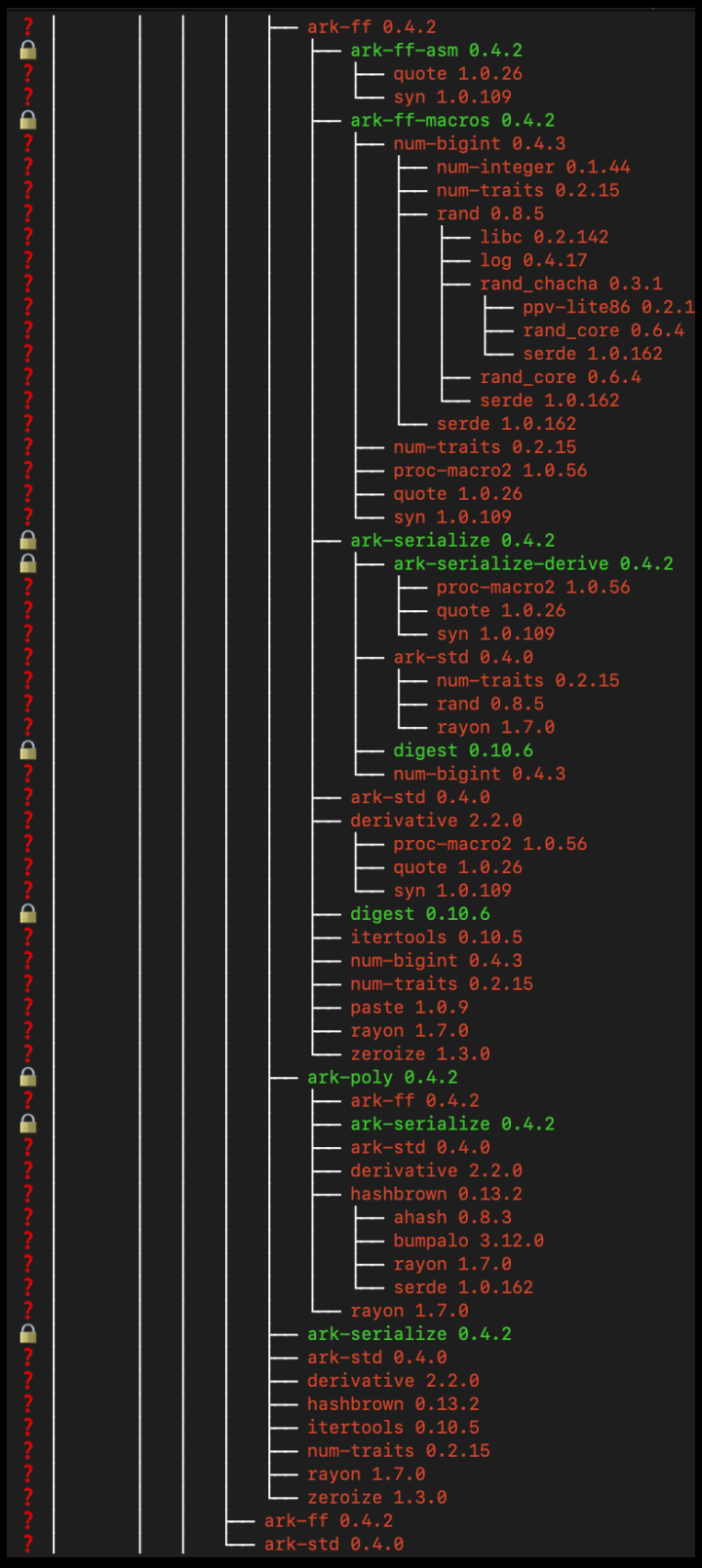
AUTOMATED TESTING



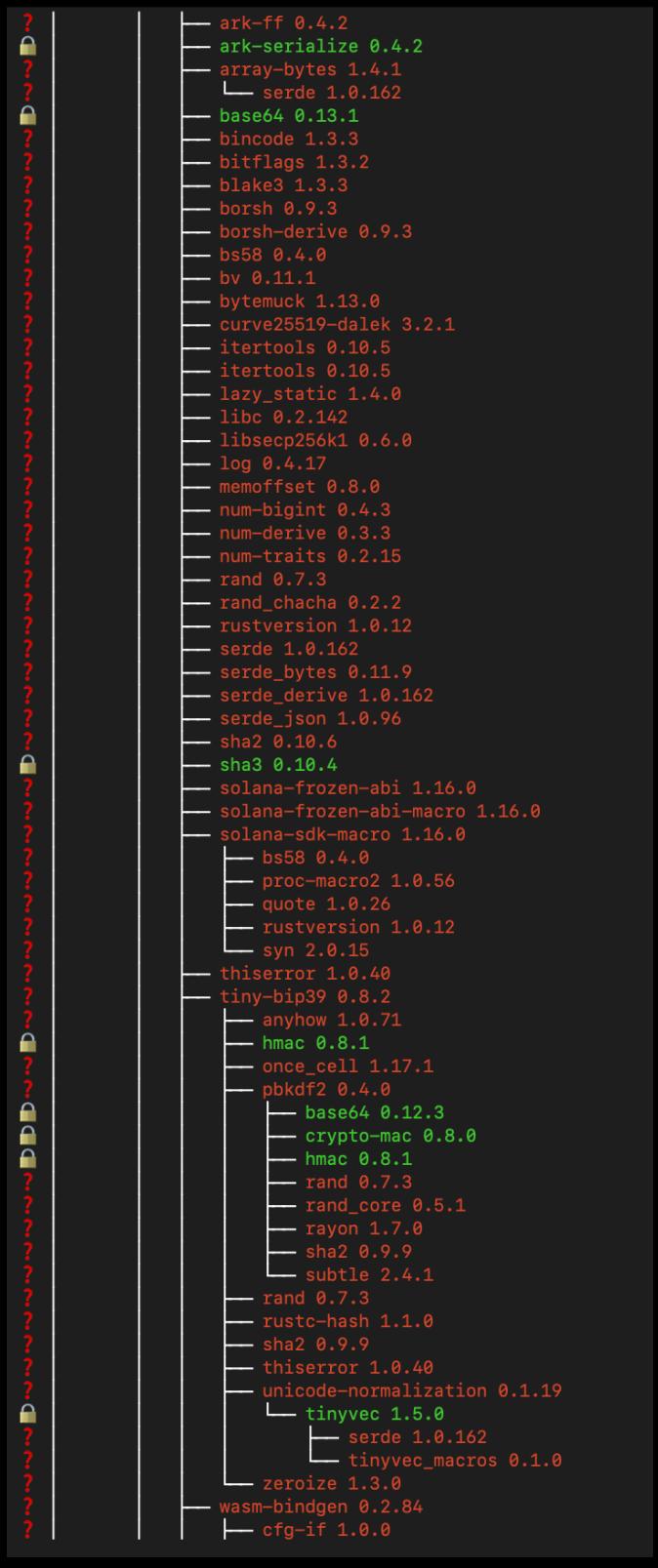
AUTOMATED TESTING



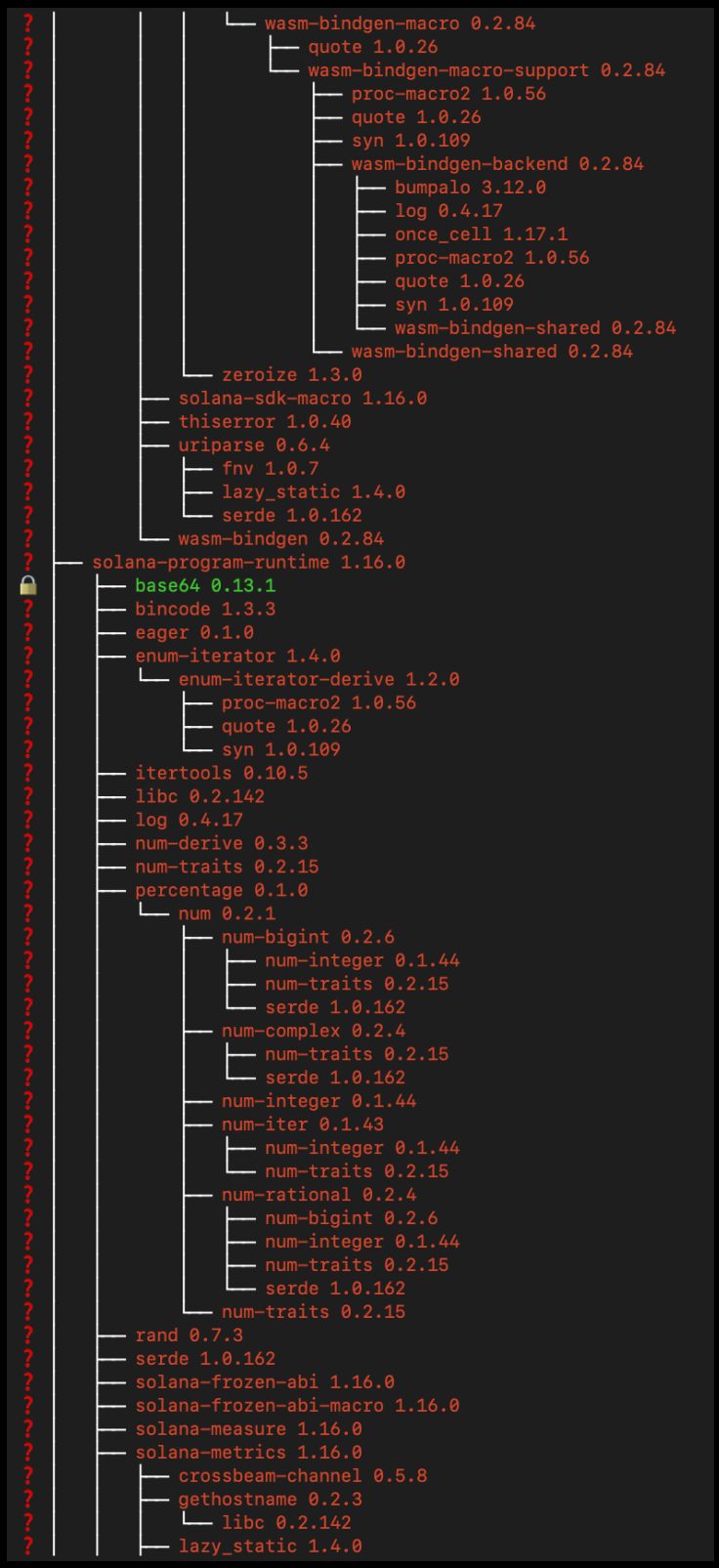
AUTOMATED TESTING



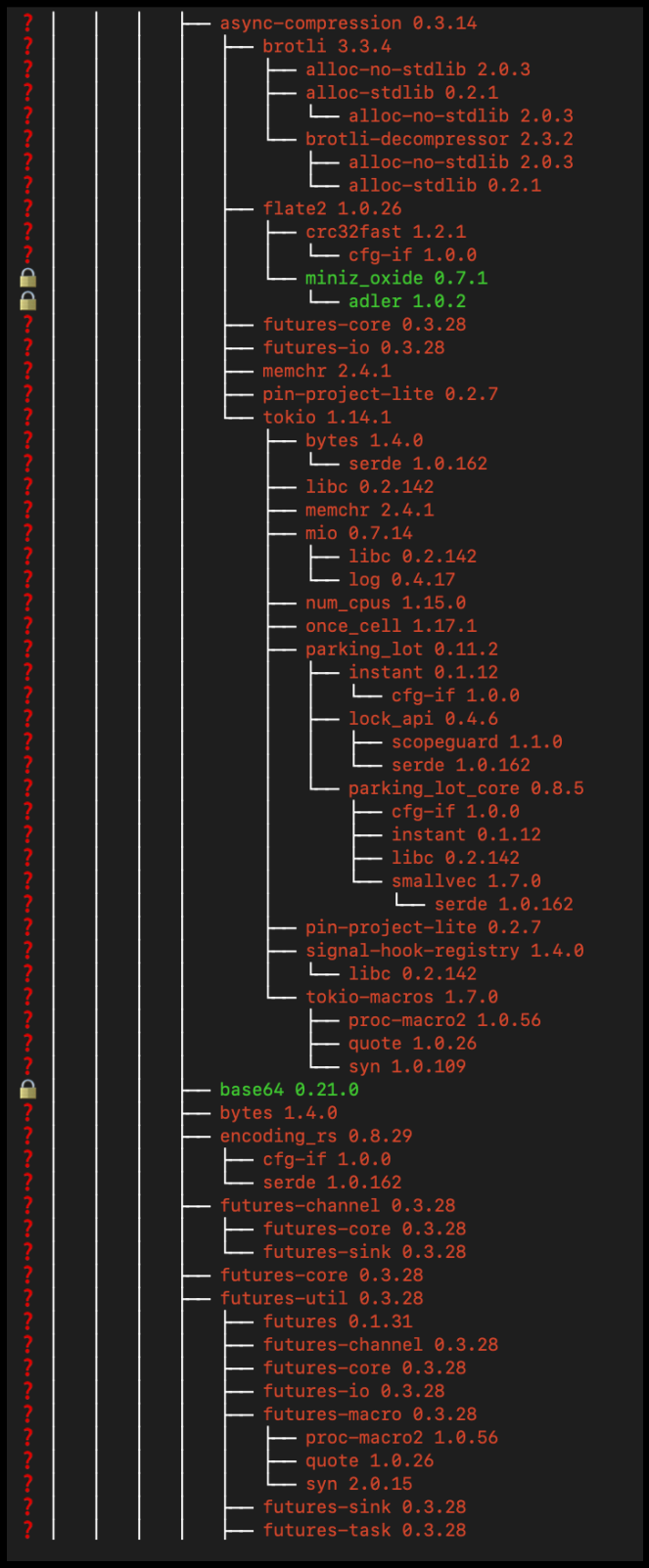
AUTOMATED TESTING



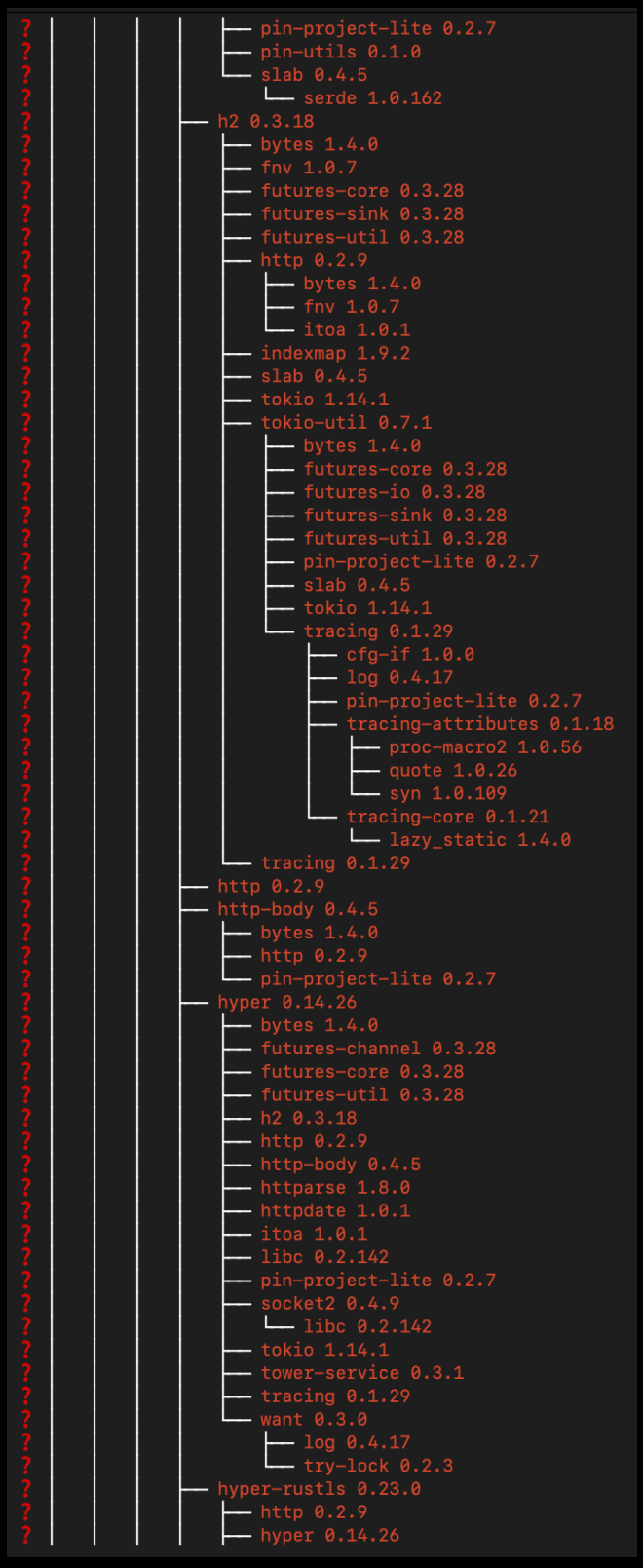
AUTOMATED TESTING



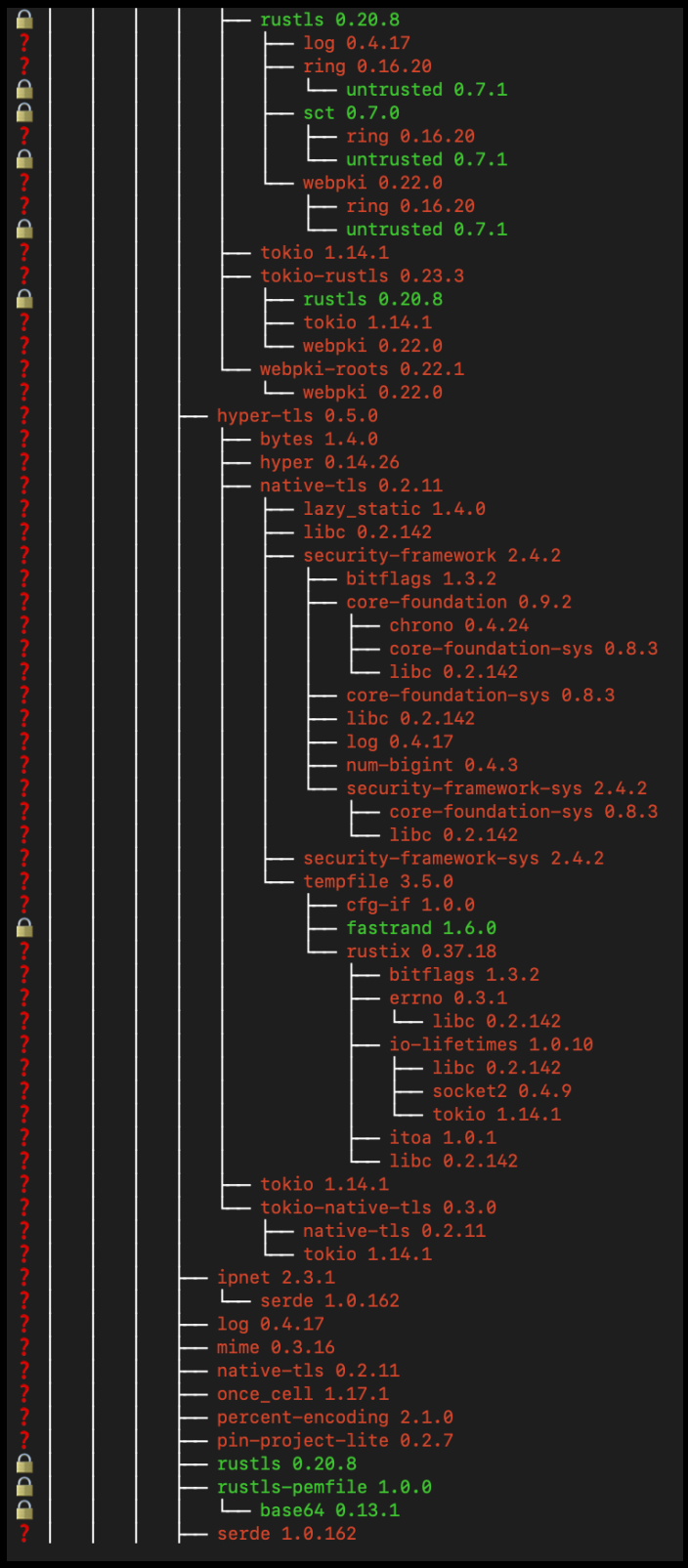
AUTOMATED TESTING



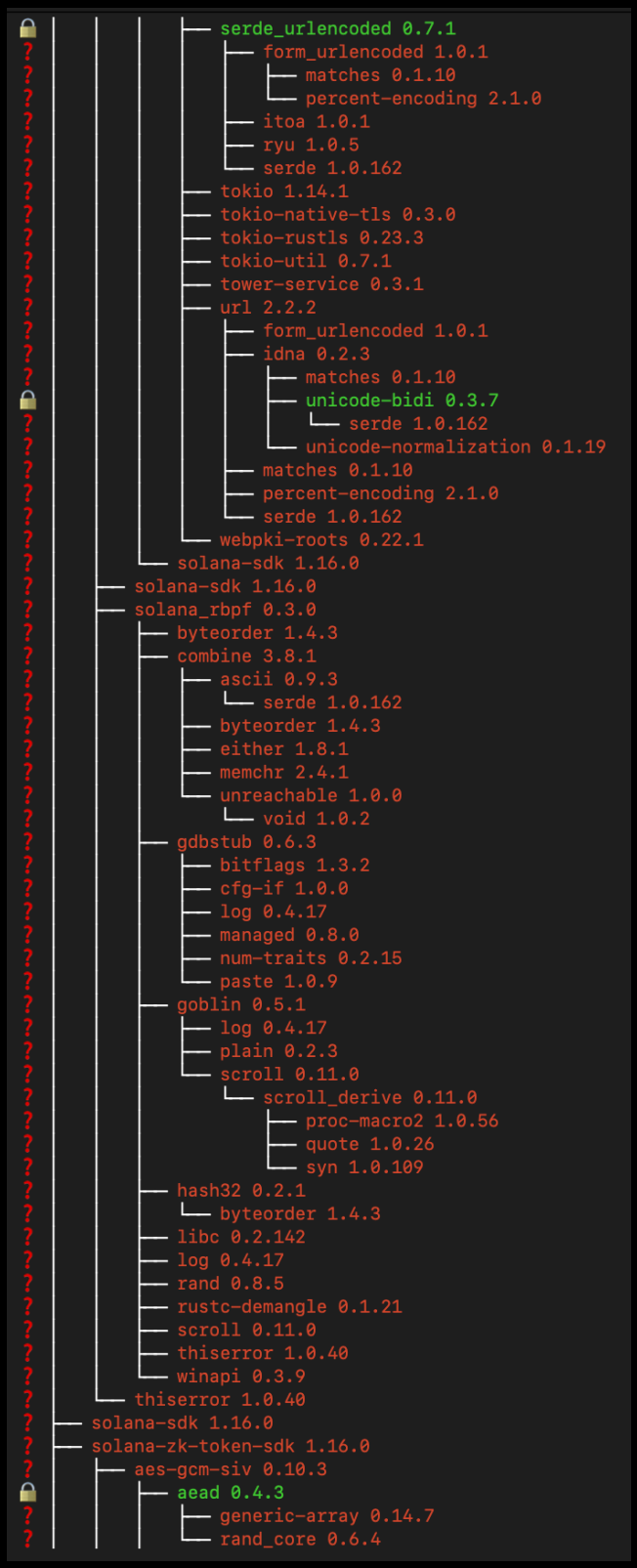
AUTOMATED TESTING



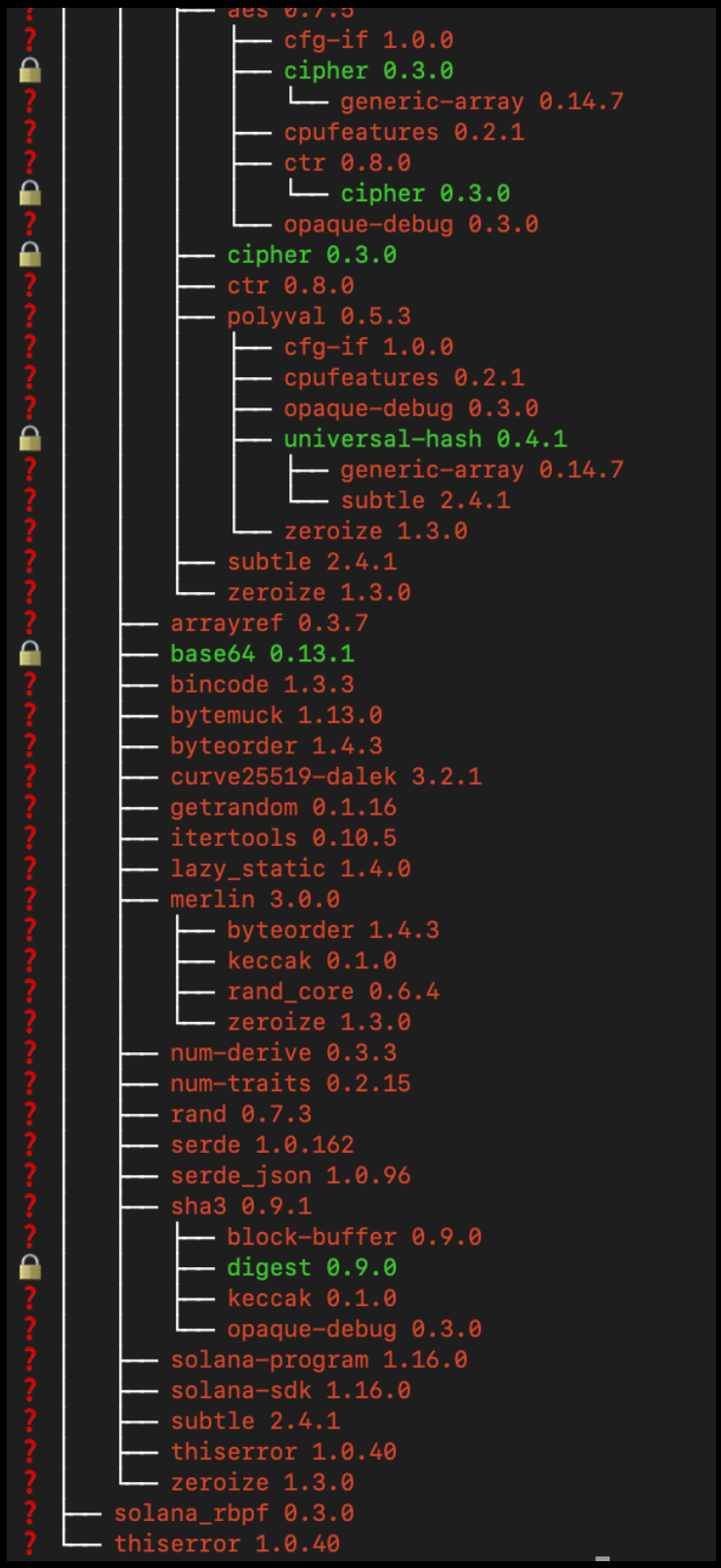
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN