HALBORN

Solana Labs - Runtime 1c862f0 -> 7cbe7d3

L1 Security Assessment

Prepared by: **Halborn**

Date of Engagement: **February 13th, 2023 - March 3rd, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 4
1 EXECUTIVE OVERVIEW 5
1.1 INTRODUCTION 6
1.2 ASSESSMENT SUMMARY 6
1.3 TEST APPROACH & METHODOLOGY 7
2 RISK METHODOLOGY 8
2.1 EXPLOITABILITY 9
2.2 IMPACT 10
2.3 SEVERITY COEFFICIENT 12
2.4 SCOPE 14
3 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 15
4 FINDINGS & TECH DETAILS 16
4.1 (HAL-01) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL(0.0) 18
Description 18
Code Location 18
BVSS 18
Recommendation 18
Remediation Plan 18
5 MANUAL TESTING 18
5.1 EPOCH ACCOUNTS HASH (EAH) 20
Description 20
Results 20
5.2 COST MODEL 21
Description 21

Results 21
5.3 PRIORITIZATION FEES 21
Description 21
Results 22
6 AUTOMATED TESTING 23
6.1 AUTOMATED ANALYSIS 24
Description 24
Results 24
6.2 UNSAFE RUST CODE DETECTION 25
Description 25
Results 26

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/28/2023	Guillermo Alvarez
0.2	Document Updates	03/01/2023	Guillermo Alvarez
0.3	Final Draft	03/02/2023	Guillermo Alvarez
0.4	Draft Review	03/02/2023	Isabel Burruezo
0.5	Draft Review	03/02/2023	Piotr Cielas
0.6	Draft Review	03/02/2023	Gabi Urrutia
1.0	Remediation Plan	08/22/2023	Isabel Burruezo
1.1	Remediation Plan Updates	08/30/2023	Isabel Burruezo
1.2	Remediation Plan Review	08/30/2023	Piotr Cielas
1.3	Remediation Plan Review	08/30/2023	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 Alvarez	Halborn	Guillermo.Alvarez@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 the Solana runtime changes introduced in commits from 1c862f0 to 7cbe7d3, beginning on February 13th, 2023 and ending on March 3rd, 2023 . The security assessment was scoped to the implementation provided in 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 3 weeks for the engagement and assigned 1 full-time security engineer to verify the security of the code in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Identify potential security issues within the programs

In summary, Halborn did not identify any significant security risk affecting the new updates introduced in the program-runtime, runtime and bpf_loader modules in scope.

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 assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage 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 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

- Repository: solana
- Commit IDs:
  - start: 1c862f0b666c6de251a630d9128313b51dfa7c46
  - final: 7cbe7d30f9082a253ca830e41a51a1dfe6570d8c
- 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	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING CARGO OVERFLOW CHECKS	Informational (0.0)	NOT APPLICABLE

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:

NOT APPLICABLE The code in scope for this assessment does not use unchecked integer arithmetic, so Solana Labs have opted not to incur potential performance penalties by enabling overflow checks.

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 code for.

5.1 EPOCH ACCOUNTS HASH (EAH)

Description:

Rent collection checks every account at least once per epoch. At each slot, a deterministic set of accounts is loaded, checked for rent collection, and stored back to the Accounts DB at this new slot. Meaning that accounts are rewritten even if they are unchanged. Since accounts are required to be rent-exempt, the majority of accounts rewritten due to rent collection are unchanged. In order to improve performance, Epoch Account Hash was introduced, once per epoch a full account's hash calculation is done and its result stored into the bank's hash, with this approach, accounts are verified at least once per epoch without the performance cut. Validators calculate the same account hash based on the same view of all accounts, to do so, there are two different offsets defined one-quarter into the epoch and three-quarters into the epoch to avoid doing the EAH calculation at the beginning of an epoch or during the stake rewards distribution, the upper-limit is defined, so validators have time to complete the calculation before the epoch ends.

Commits c5c58a2 and 5c3bb86 added EpochAccountsHash to AccountsDB and to bank's hash. Tests were performed to ensure that EAH hashes are properly added into the bank's hash each epoch. Additionally, it was tested that EAH is not calculated if epochs do not have the default 432,000 like during warm up as defined in the implementation proposal and if this could be bypassed.

Results:

No code vulnerabilities were identified.

5.2 COST MODEL

Description:

Several changes were applied to the transaction Cost Model, instead of using estimated BPF instruction costs now compute_budget.set_compute_unit_limit is used to keep track of BPF costs. Most of the changes were implemented in commits c8b0c3e, 6c1dc69 and 6f4fe37. It was reviewed that built-in programs have a predefined cost also tests for checking if different costs for the same transaction, which could happen with validators running old clients, would have an impact in the consensus mechanism or in the transaction processing.

Results:

No code vulnerabilities were identified.

5.3 PRIORITIZATION FEES

Description:

Additional fees were introduced to transactions as a method to allow users to bid for priority for their transactions in the leader's queue. Priority fees allow users to have more control over when transactions will be confirmed even at times when the network is congested, if the fee is high enough, the transaction should land in a block. However, the initial implementation did not provide a method to estimate the priority fee that a user should pay to ensure that their transaction would be executed, in order to keep track of the block minimum prioritization fee, a new struct called PrioritizationFeeCache was added, this struct keeps track of the latest 150 blocks associated fees.
Commits 8bb039d and 0244b01 introduced the changes, which were reviewed to ensure that they can be used to estimate priority fees, and they cannot be bypassed to arbitrarily increase prioritization fees affecting users.

Results:

No code vulnerabilities were identified.

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
RUSTSEC-2021-0139	ansi_term	ansi_term is unmaintained
RUSTSEC-2020-0016	net2	net2 crate has been deprecated

6.2 UNSAFE RUST CODE DETECTION

Description:

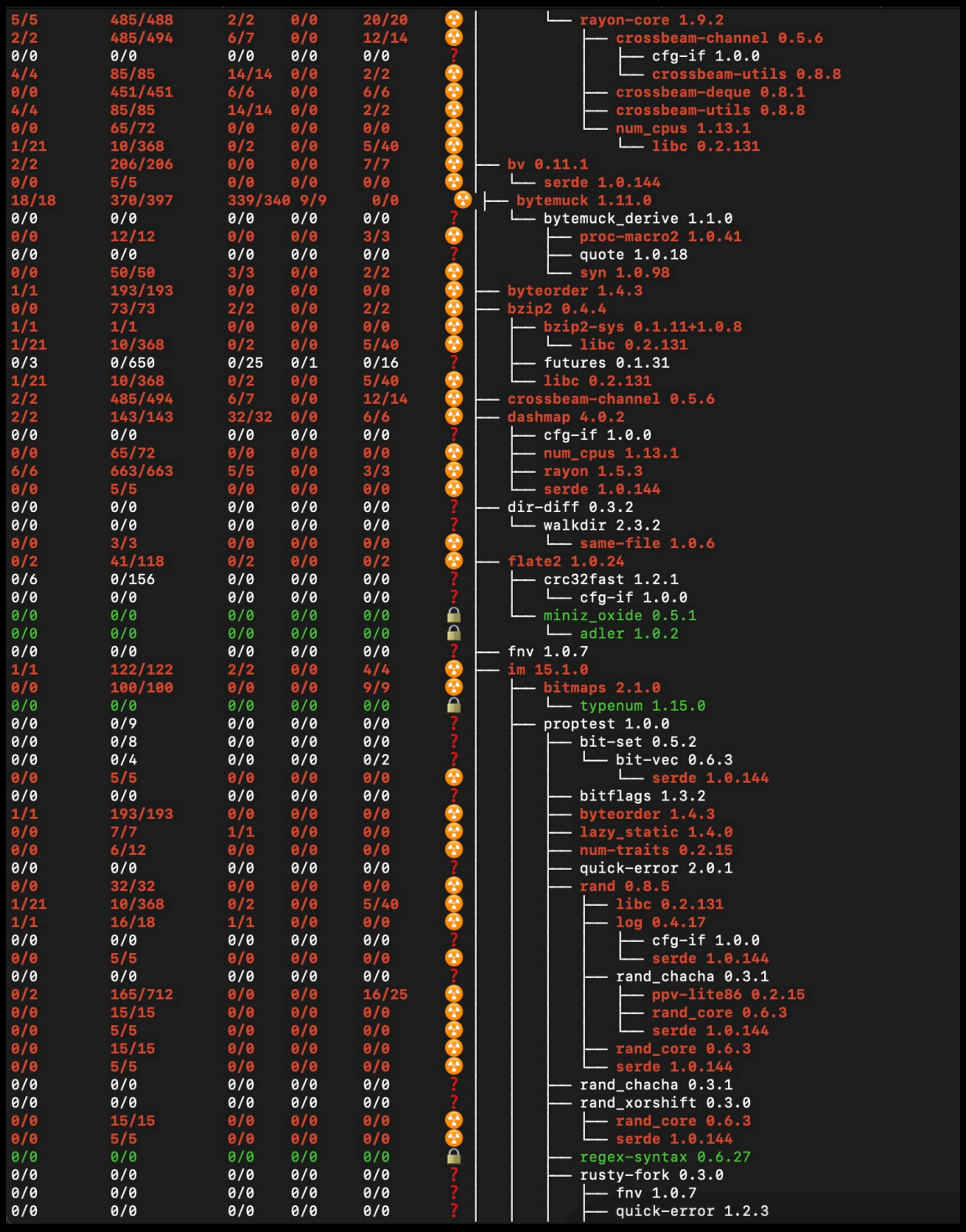
Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

AUTOMATED TESTING

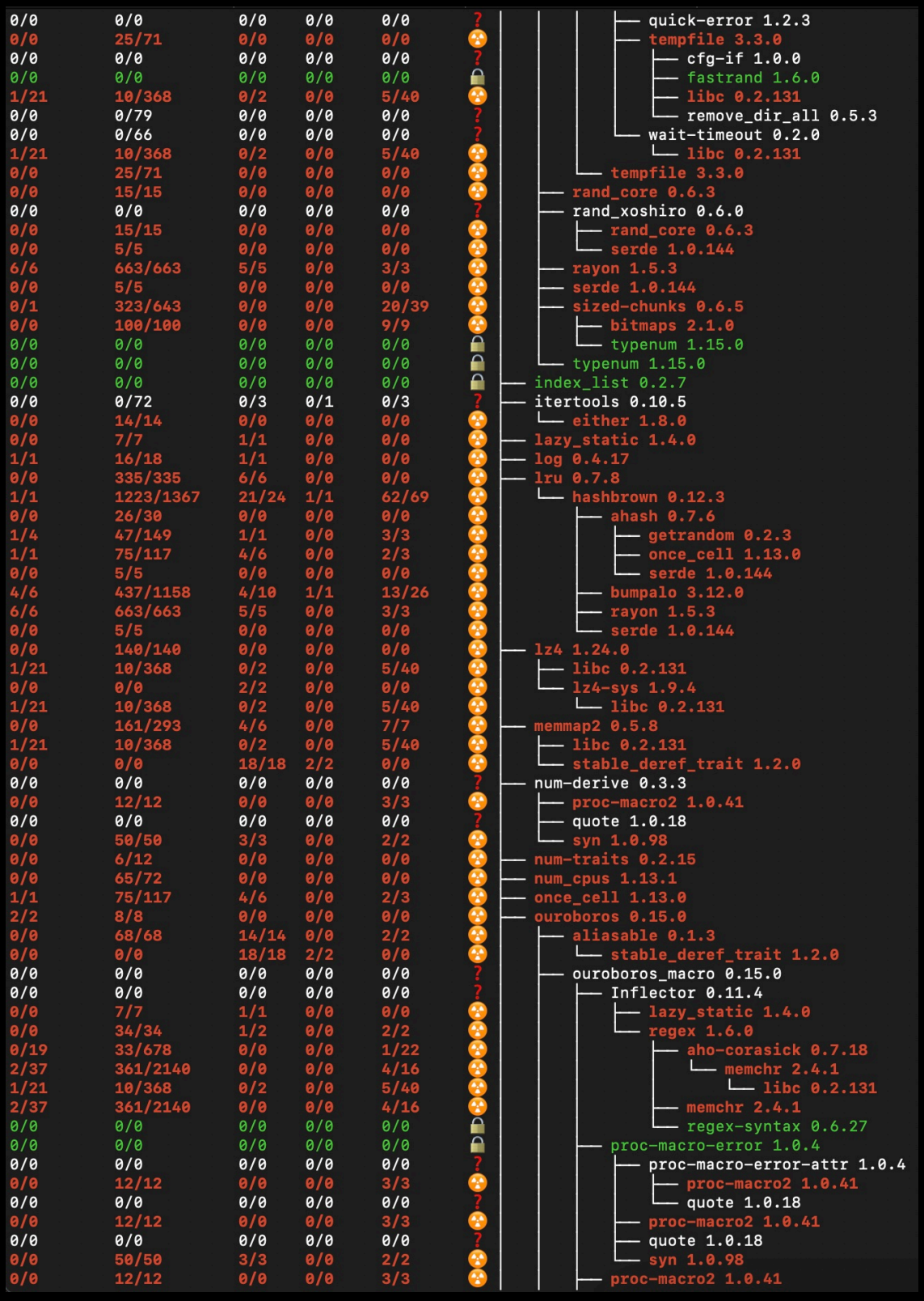
Results:

```
Symbols:
    🔒 = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓ = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢ = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        46/46        0/0    0/0     0/0      ☢ solana-runtime 1.15.1
0/0        0/0          0/0    0/0     0/0      ❓ ├── arrayref 0.3.6
0/0        22/22        0/0    0/0     0/0      ☢ ├── bincode 1.3.3
0/0        5/5          0/0    0/0     0/0      ☢ │   └── serde 1.0.144
0/0        0/0          0/0    0/0     0/0      ❓ │       └── serde_derive 1.0.144
0/0        12/12        0/0    0/0     3/3      ☢ │           ├── proc-macro2 1.0.41
0/0        4/4          0/0    0/0     0/0      ☢ │           │   └── unicode-ident 1.0.2
0/0        0/0          0/0    0/0     0/0      ❓ │           ├── quote 1.0.18
0/0        12/12        0/0    0/0     3/3      ☢ │           │   └── proc-macro2 1.0.41
0/0        50/50        3/3    0/0     2/2      ☢ │           └── syn 1.0.98
0/0        12/12        0/0    0/0     3/3      ☢ │               ├── proc-macro2 1.0.41
0/0        0/0          0/0    0/0     0/0      ❓ │               ├── quote 1.0.18
0/0        4/4          0/0    0/0     0/0      ☢ │               └── unicode-ident 1.0.2
2/78       29/3973      0/0    0/0     0/0      ☢ ├── blake3 1.3.1
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── arrayref 0.3.6
2/2        350/350      2/2    0/0     7/7      ☢ │   ├── arrayvec 0.7.2
0/0        5/5          0/0    0/0     0/0      ☢ │   │   └── serde 1.0.144
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── cfg-if 1.0.0
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── constant_time_eq 0.1.5
0/0        0/0          0/0    0/0     0/0      🔒 │   ├── digest 0.10.3
0/0        16/16        0/0    0/0     0/0      ☢ │   │   ├── block-buffer 0.10.2
1/1        285/285      20/20  8/8     5/5      ☢ │   │   │   └── generic-array 0.14.6
0/0        5/5          0/0    0/0     0/0      ☢ │   │   │       ├── serde 1.0.144
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │       ├── typenum 1.15.0
1/1        23/23        0/0    0/0     0/0      ☢ │   │   │       └── zeroize 1.3.0
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │           └── zeroize_derive 1.2.0
0/0        12/12        0/0    0/0     3/3      ☢ │   │   │               ├── proc-macro2 1.0.41
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │               ├── quote 1.0.18
0/0        50/50        3/3    0/0     2/2      ☢ │   │   │               ├── syn 1.0.98
0/0        0/0          1/1    0/0     0/0      ☢ │   │   │               └── synstructure 0.12.6
0/0        12/12        0/0    0/0     3/3      ☢ │   │   │                   ├── proc-macro2 1.0.41
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │                   ├── quote 1.0.18
0/0        50/50        3/3    0/0     2/2      ☢ │   │   │                   ├── syn 1.0.98
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │                   └── unicode-xid 0.2.2
0/0        0/0          0/0    0/0     0/0      🔒 │   │   ├── crypto-common 0.1.3
1/1        285/285      20/20  8/8     5/5      ☢ │   │   │   ├── generic-array 0.14.6
0/0        15/15        0/0    0/0     0/0      ☢ │   │   │   ├── rand_core 0.6.3
1/4        47/149       1/1    0/0     3/3      ☢ │   │   │   │   ├── getrandom 0.2.3
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   ├── cfg-if 1.0.0
1/21       10/368       0/2    0/0     5/40     ☢ │   │   │   │   │   └── libc 0.2.131
0/0        5/5          0/0    0/0     0/0      ☢ │   │   │   │   └── serde 1.0.144
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │   └── typenum 1.15.0
0/0        3/3          0/0    0/0     0/0      ☢ │   │   └── subtle 2.4.1
6/6        663/663      5/5    0/0     3/3      ☢ │   └── rayon 1.5.3
0/0        451/451      6/6    0/0     6/6      ☢ │       ├── crossbeam-deque 0.8.1
0/0        0/0          0/0    0/0     0/0      ❓ │       │   ├── cfg-if 1.0.0
3/3        421/433      9/9    0/0     26/26    ☢ │       │   ├── crossbeam-epoch 0.9.5
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │   ├── cfg-if 1.0.0
4/4        85/85        14/14  0/0     2/2      ☢ │       │   │   ├── crossbeam-utils 0.8.8
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │   │   ├── cfg-if 1.0.0
0/0        7/7          1/1    0/0     0/0      ☢ │       │   │   │   └── lazy_static 1.4.0
0/0        0/49         0/6    0/0     0/3      ❓ │       │   │   │       └── spin 0.5.2
0/0        7/7          1/1    0/0     0/0      ☢ │       │   │   ├── lazy_static 1.4.0
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │   ├── memoffset 0.6.4
0/0        18/18        1/1    0/0     0/0      ☢ │       │   │   └── scopeguard 1.1.0
4/4        85/85        14/14  0/0     2/2      ☢ │       │   └── crossbeam-utils 0.8.8
0/0        14/14        0/0    0/0     0/0      ☢ │       ├── either 1.8.0
0/0        5/5          0/0    0/0     0/0      ☢ │       │   └── serde 1.0.144
5/5        485/488      2/2    0/0     20/20    ☢ │       └── rayon-core 1.9.2
```

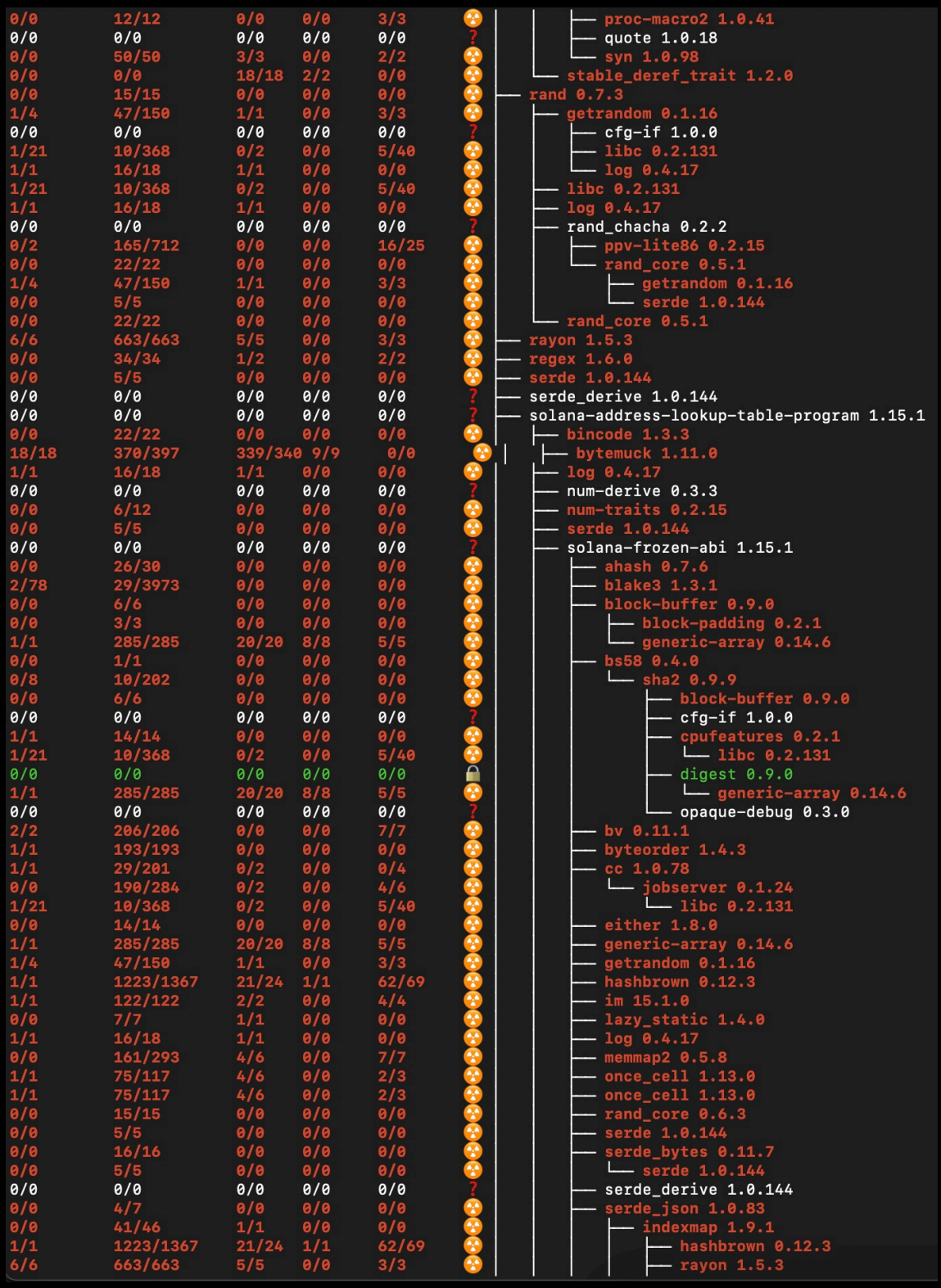
AUTOMATED TESTING



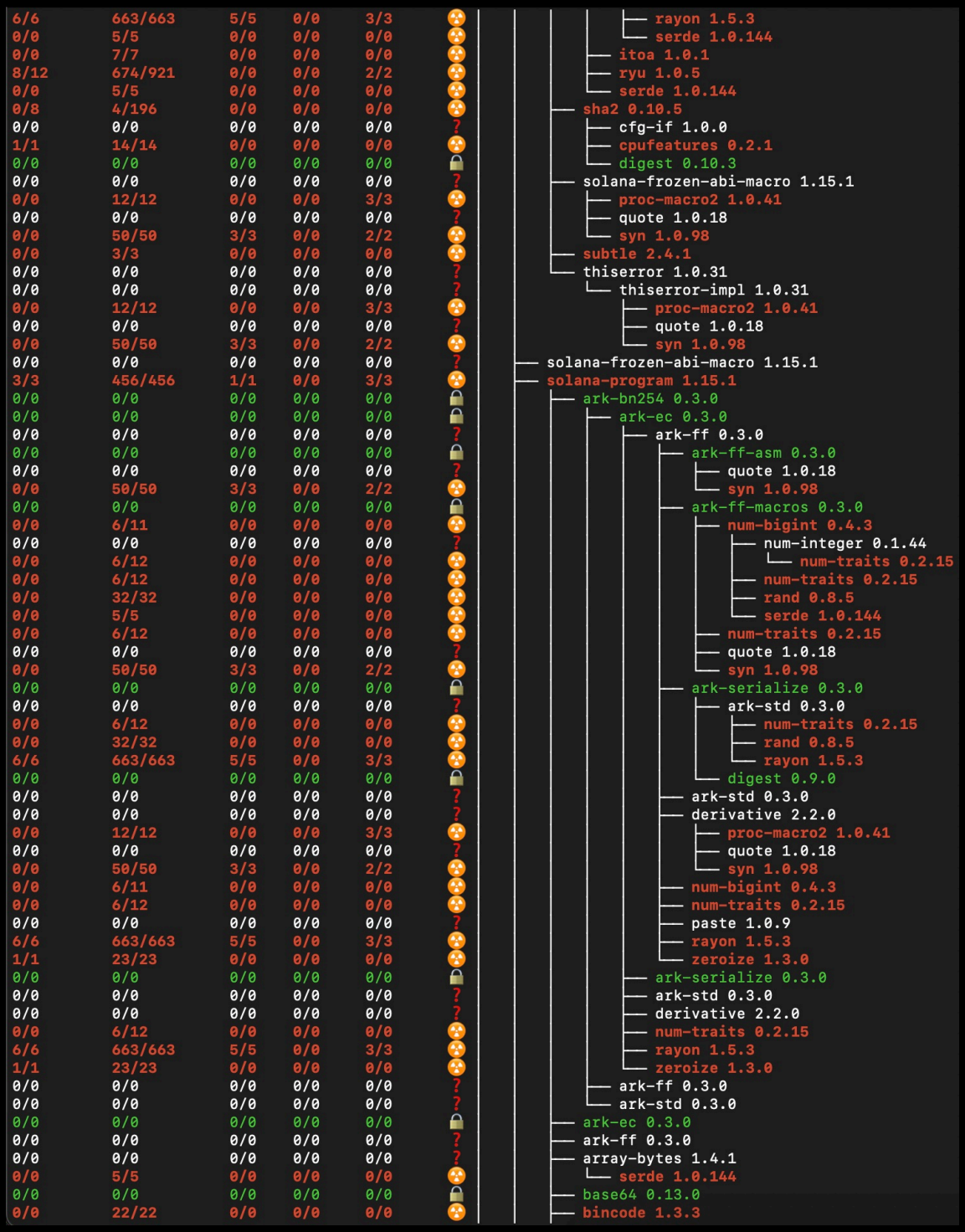
AUTOMATED TESTING



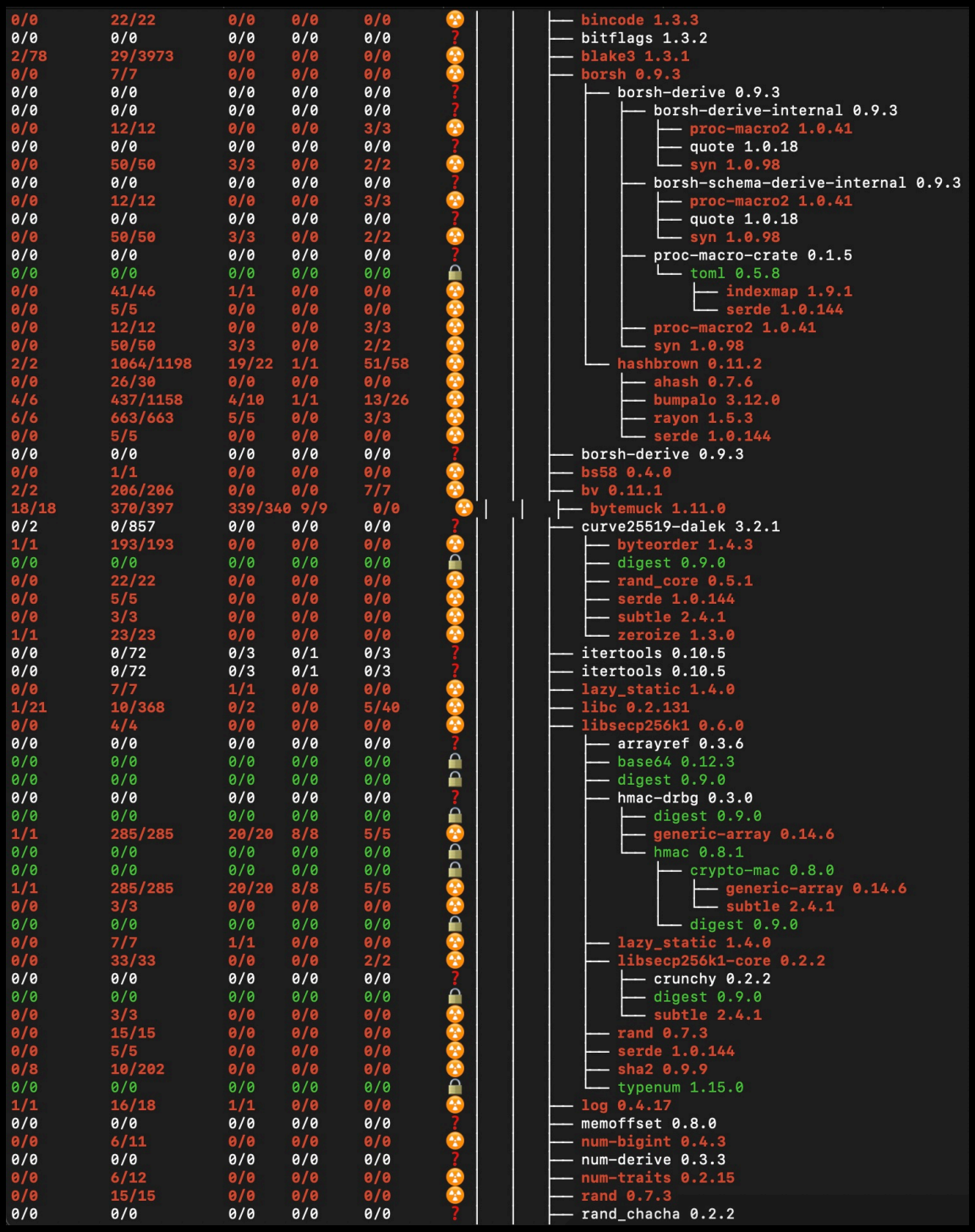
AUTOMATED TESTING



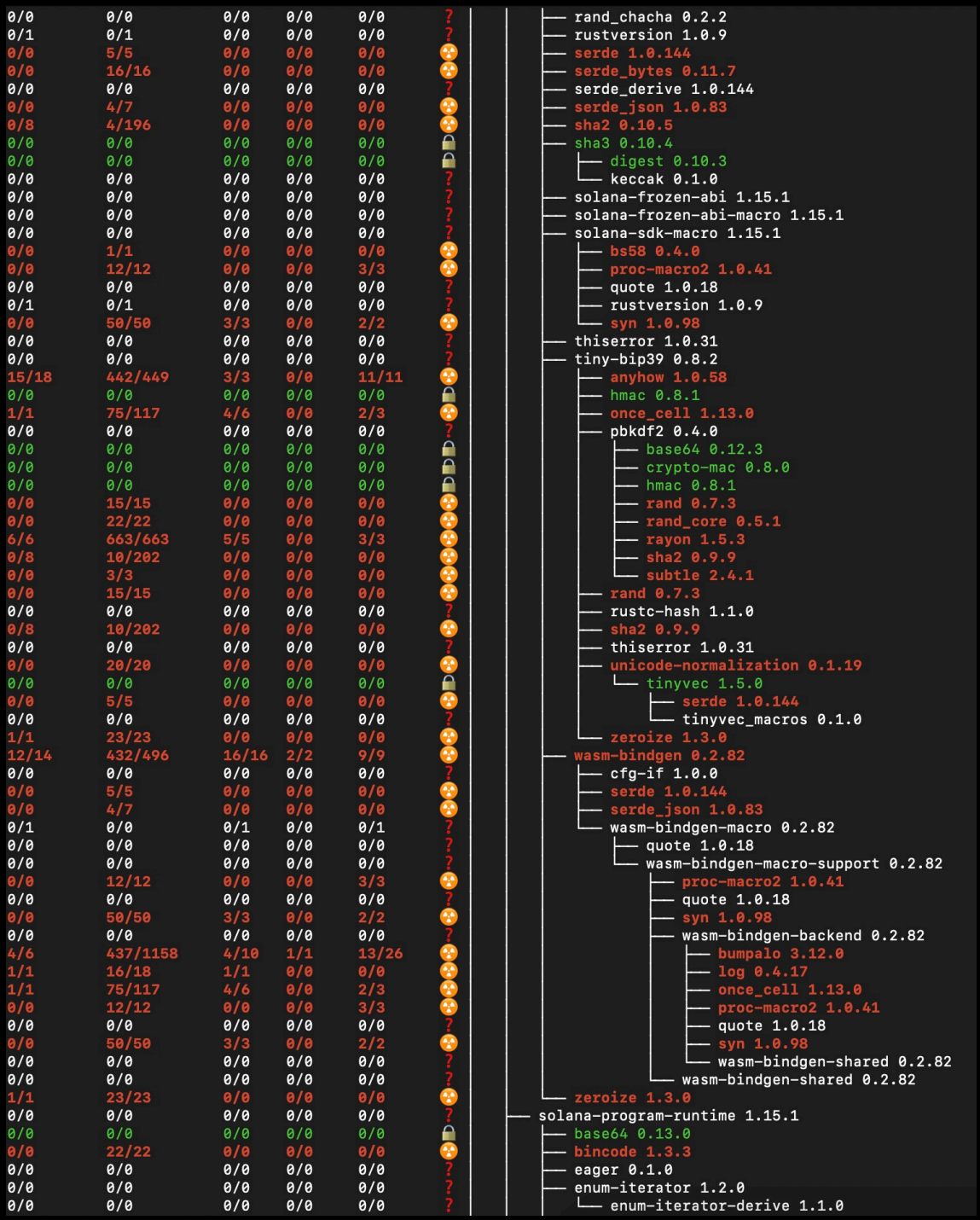
AUTOMATED TESTING



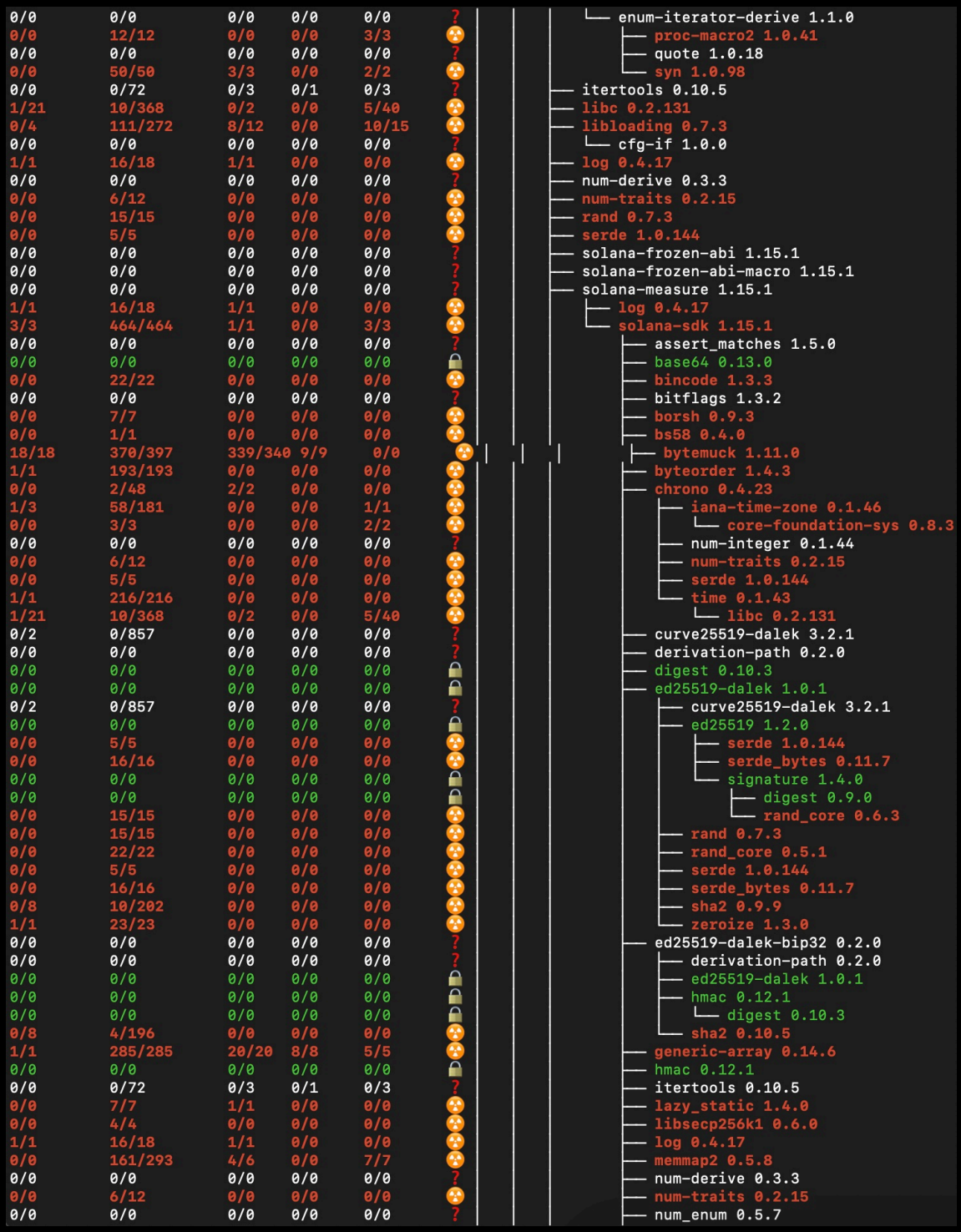
AUTOMATED TESTING



AUTOMATED TESTING

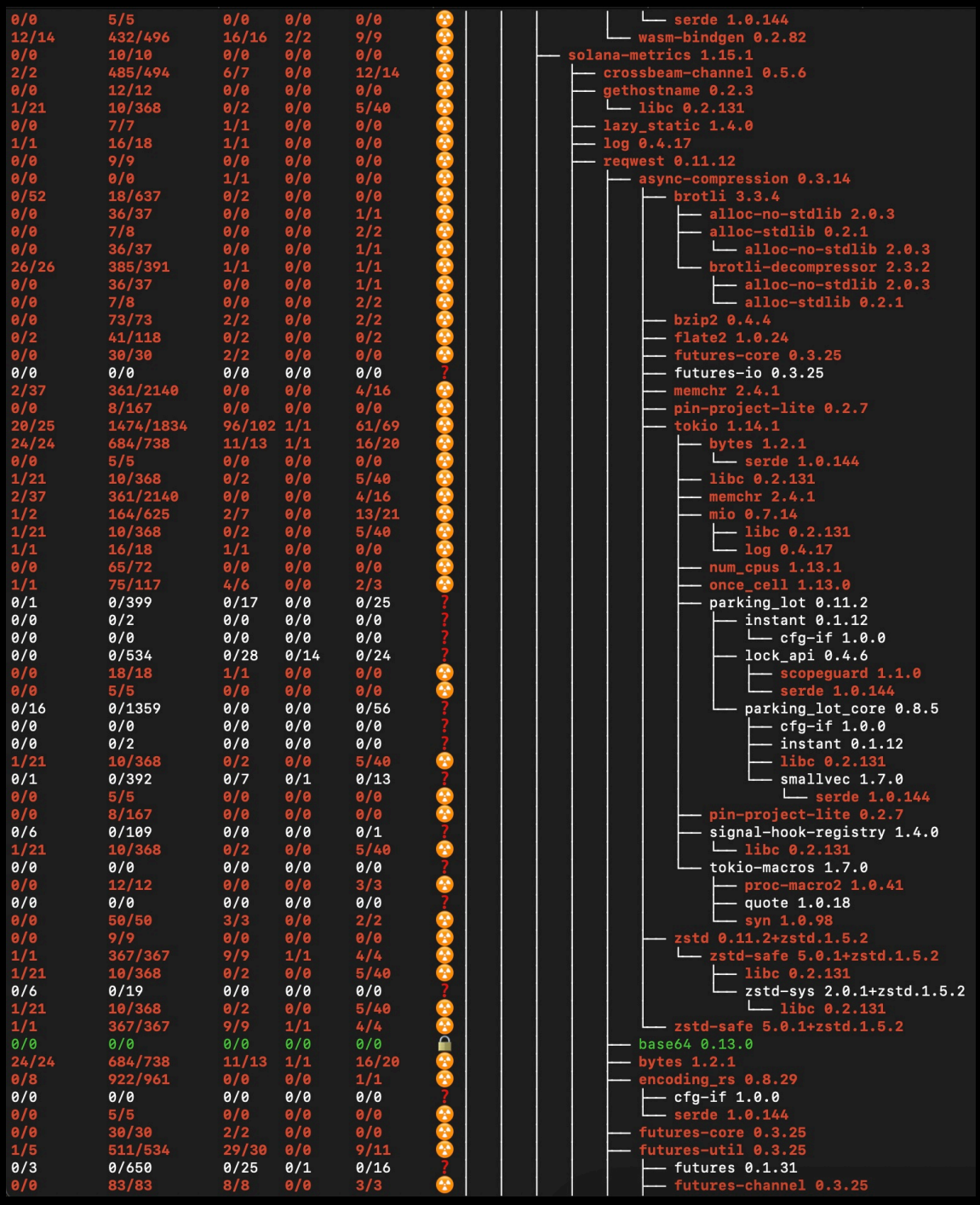


AUTOMATED TESTING

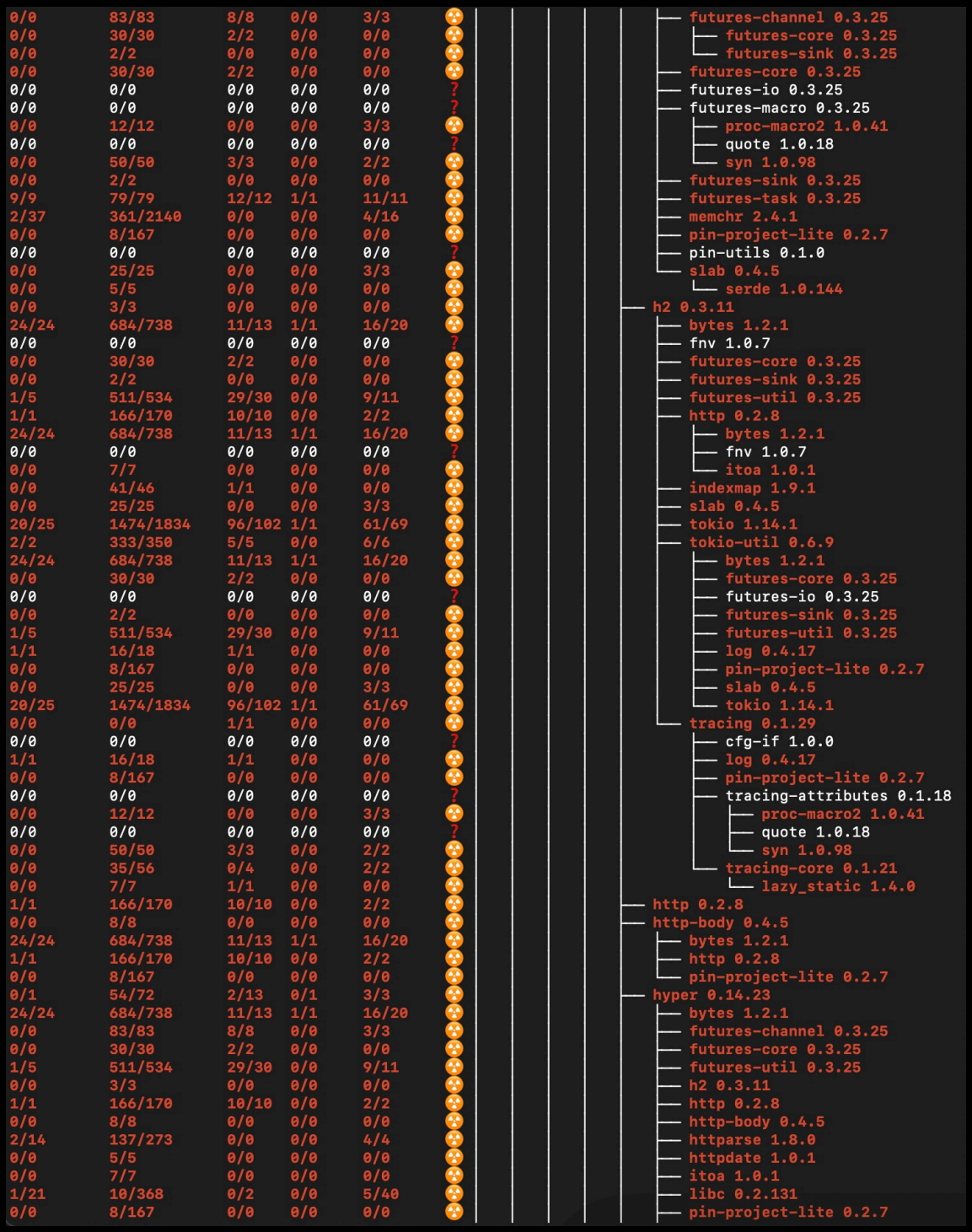


AUTOMATED TESTING

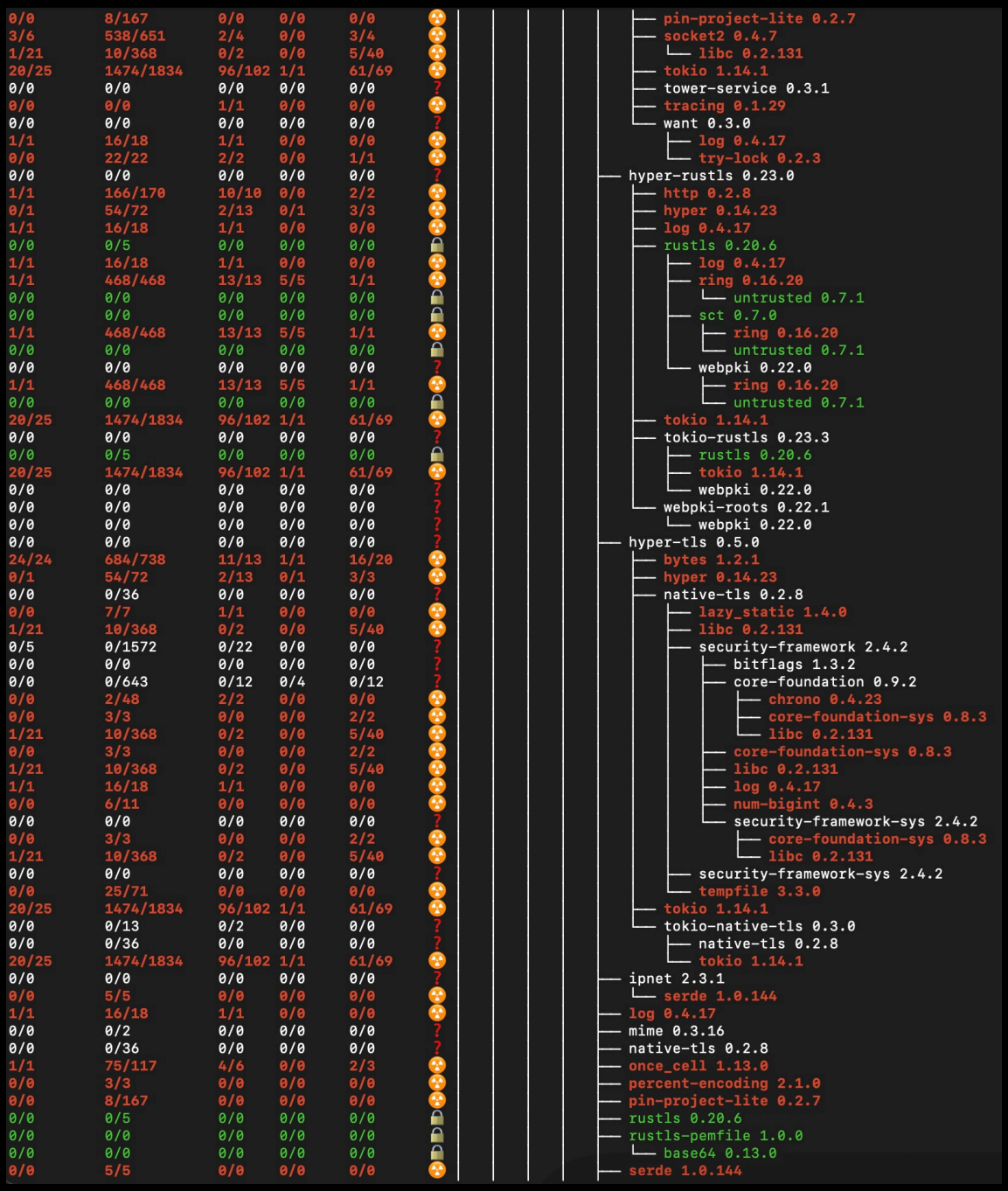
AUTOMATED TESTING



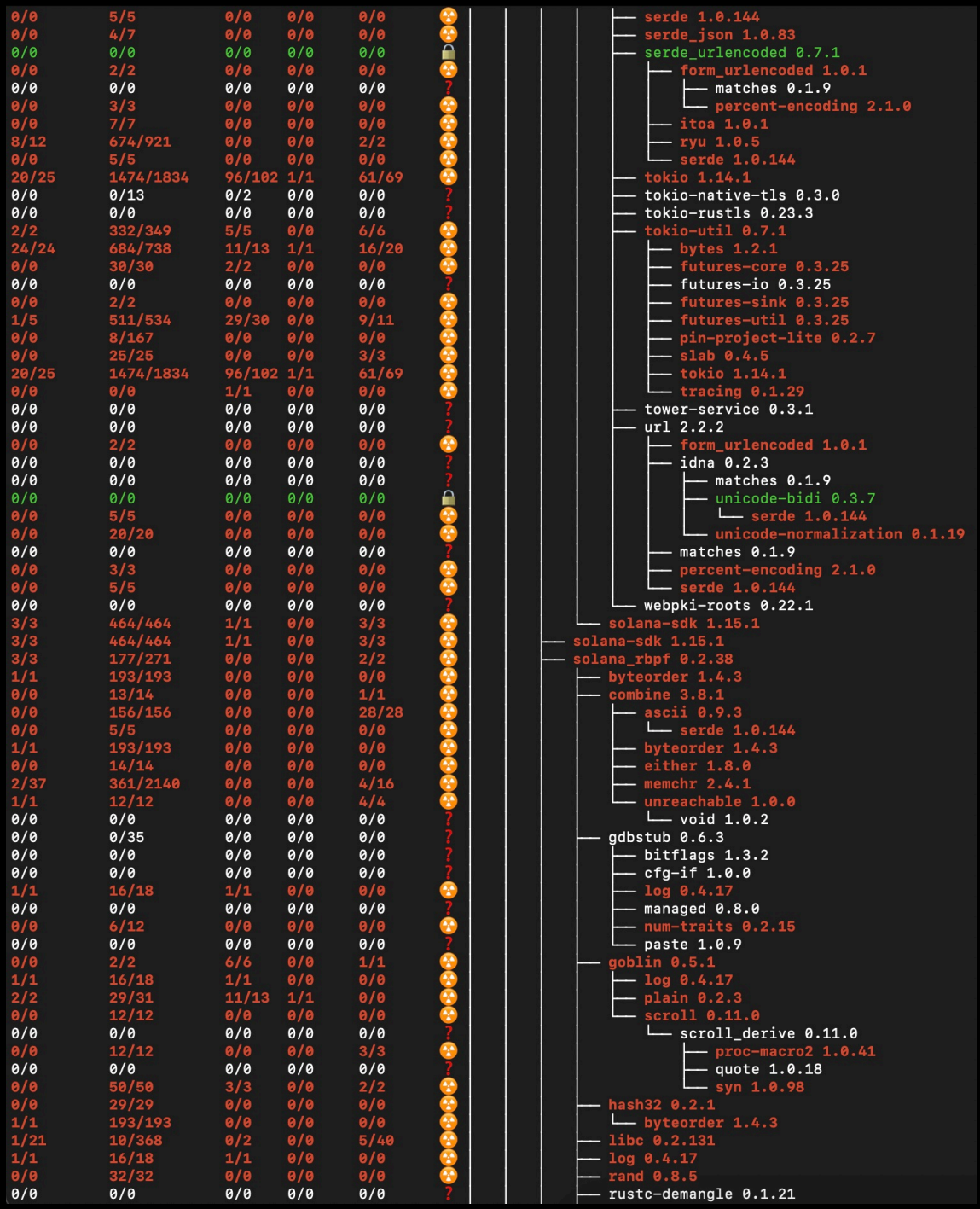
AUTOMATED TESTING



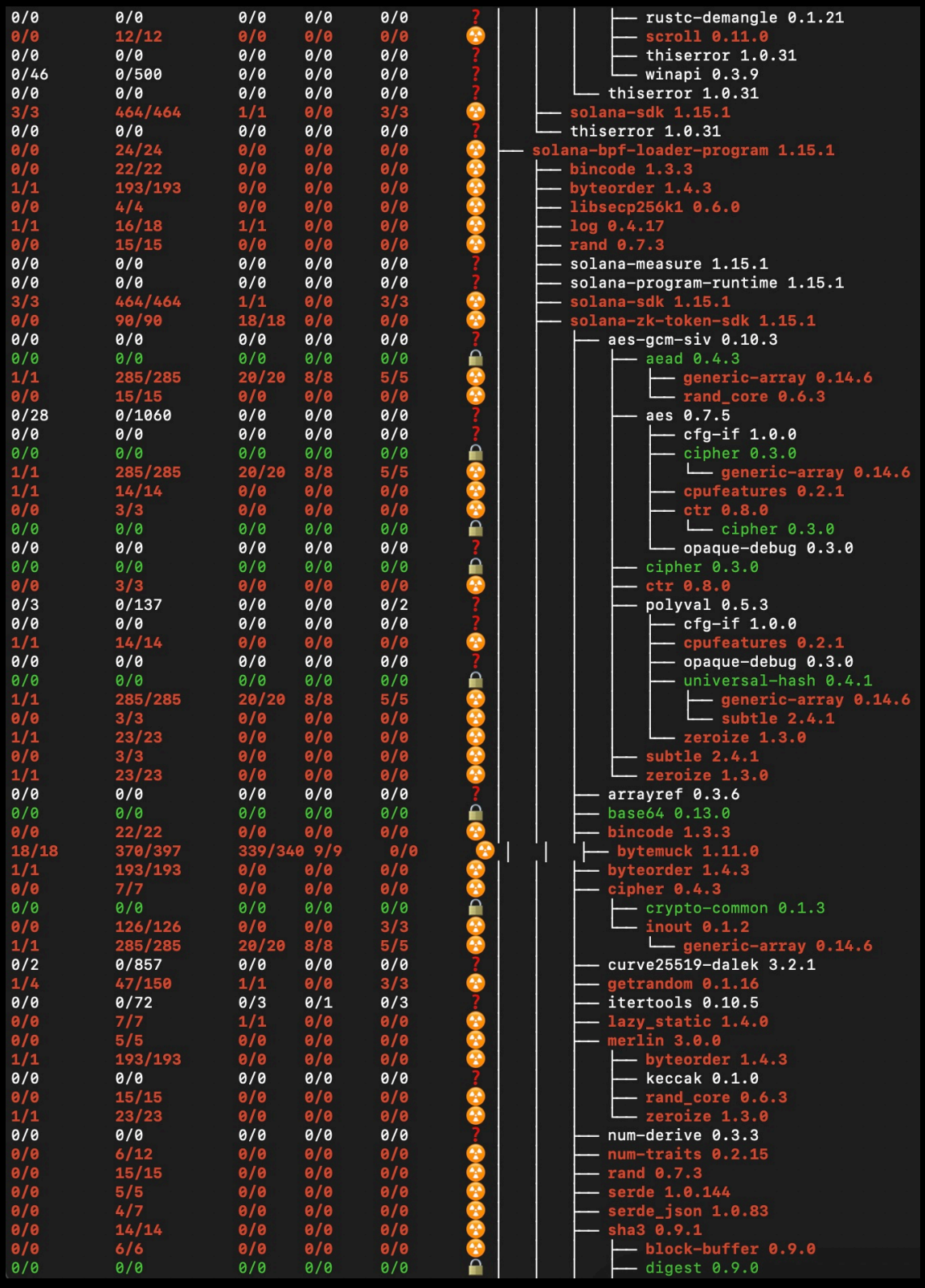
AUTOMATED TESTING



AUTOMATED TESTING

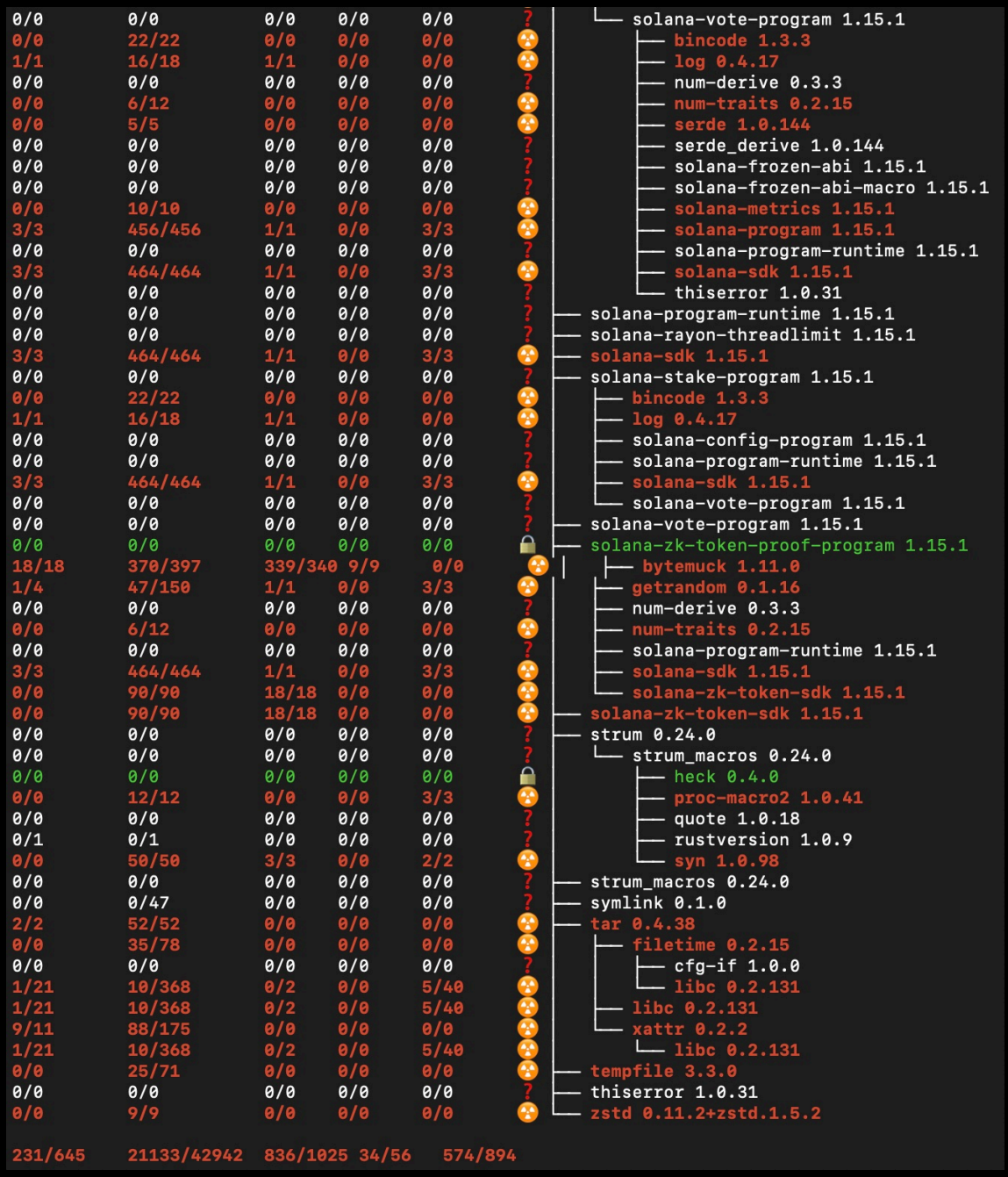


AUTOMATED TESTING



AUTOMATED TESTING

AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN