HALBORN

Solana Labs - Solana v1.14.1 - v1.14.6

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 23rd, 2022 - December 9th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/02/2022	Guillermo Alvarez
0.2	Document Updates	12/08/2022	Michael Smith
0.3	Final Draft	12/08/2022	Guillermo Alvarez
0.4	Draft Review	12/09/2022	Piotr Cielas
0.5	Draft Review	12/09/2022	Gabi Urrutia
1.0	Remediation Plan	01/05/2023	Guillermo Alvarez
1.1	Remediation Plan Review	01/05/2023	Isabel Burruezo
1.2	Remediation Plan Review	01/05/2023	Piotr Cielas
1.3	Remediation Plan Review	01/05/2023	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
Michaell 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 is able to 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 audit on the Solana v1.14.1 to v1.14.6 changes beginning on November 23rd, 2022 and ending on December 9th, 2022 . The security assessment was scoped to the implementation of the updates up to v1.14.6 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 2 weeks for the engagement and assigned 1 full-time security engineer to audit the security of the programs 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 audit is to:

- Identify potential security issues within the programs

In summary, Halborn did not identify any significant security risk affecting the new updates introduced from version 1.14.1 to version 1.14.6. An informational finding was presented and acknowledged by the Solana 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 audit. 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 audit:

- 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)

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:

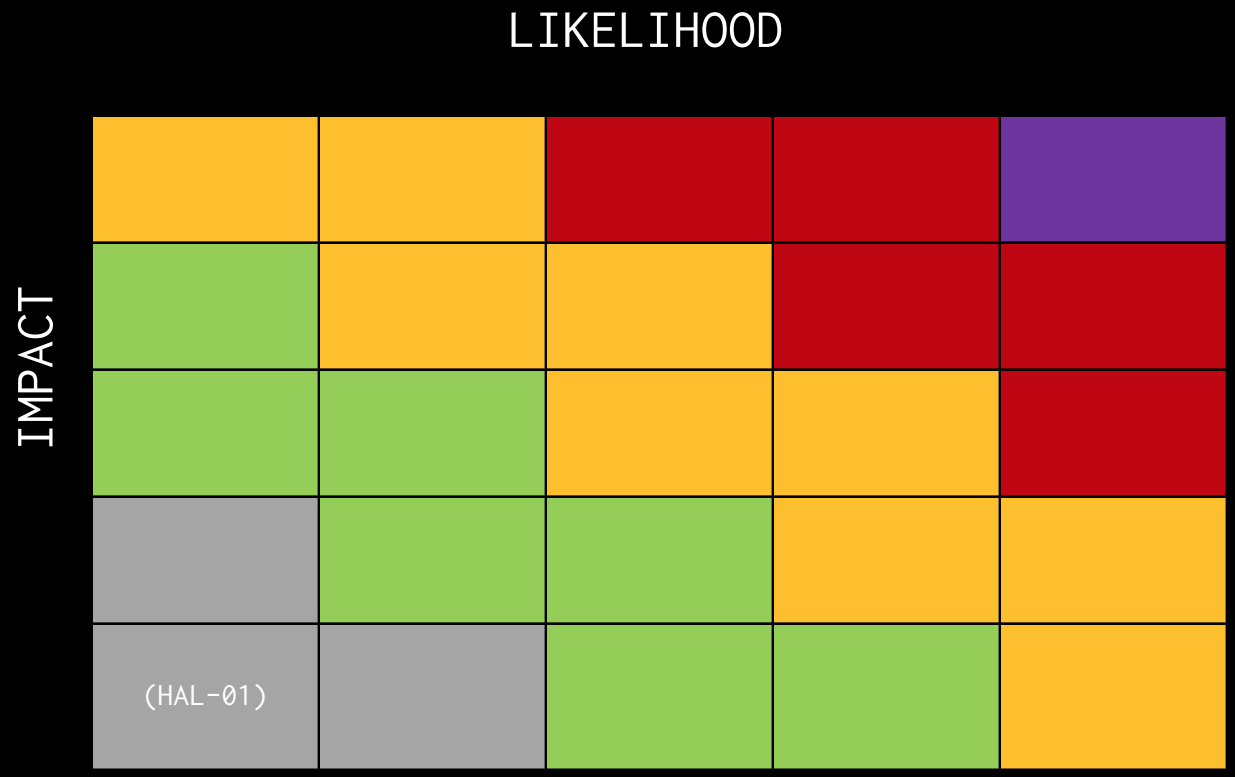
1. Solana

- Repository: solana
- Commits v1.14.1-v1.14.6:
  - start: 8aec9795bcfc0a67226a2f7b1aaeee9231b827cf
  - end: cfb2cbe1edb33faf3f8a5d9d053b531d0622f077

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL

Description:

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

Code Location:

Note: only unwraps introduced by the changes in scope are listed, justified usages such as in tests were excluded.

Listing 1

```
./core/src/ancestor_hashes_service.rs:643
↳ repairable_dead_slot_pool.take(&slot).unwrap() ;
./core/src/broadcast_stage/broadcast_fake_shreds_run.rs:143
↳ sock.send_to(b.payload(), peer.tvu_forwards).unwrap() ;
./core/src/broadcast_stage/standard_broadcast_run.rs:76
↳                         Shredder::new(state.slot, state.parent,
↳ reference_tick, self.shred_version).unwrap() ;
./core/src/broadcast_stage/standard_broadcast_run.rs:133
↳             Shredder::new(slot, parent_slot, reference_tick, self.
↳ shred_version).unwrap() ;
./core/src/repair_service.rs:274      let root_bank = repair_info.
↳ bank_forks.read().unwrap() .root_bank();
./core/src/serve_repair.rs:456     let root_bank = repair_info.
↳ bank_forks.read().unwrap() .root_bank();
./core/src/snapshot_packager_service.rs:301     let
↳ link_snapshots_dir = tempfile::tempdir_in(temp_dir).unwrap() ;
./core/src/snapshot_packager_service.rs:303         let
↳ snapshot_file_name = snapshots_path.file_name().unwrap() ;
```

```
   ./core/src/tower_storage.rs:176     fs::create_dir_all(filename.
↳ parent().unwrap() )?;
   ./core/src/validator.rs:1597          *self.start_progress.write()
↳ .unwrap()  = ValidatorStartProgress::LoadingLedger;
   ./core/src/window_service.rs:415         .unwrap() ;
   ./gossip/src/cluster_info.rs:471       self.my_contact_info.write
↳ ().unwrap() .wallclock = now;
   ./gossip/src/cluster_info.rs:481           .unwrap()
   ./gossip/src/cluster_info.rs:492    let ContactInfo {
↳ shred_version, .. } = *self.my_contact_info.read().unwrap() ;
   ./gossip/src/cluster_info.rs:3348     .map(|(ping, (keypair,
↳ socket))| (*socket, Pong::new(ping, keypair).unwrap() ))
   ./gossip/src/cluster_info.rs:3391     .map(|ping| Pong::new(ping,
↳ &this_node).unwrap() )
   ./gossip/src/cluster_info.rs:3402     .unwrap() ;
   ./gossip/src/cluster_info.rs:4018        let mut node =
↳ cluster_info.my_contact_info.write().unwrap() ;
   ./gossip/src/crds_gossip.rs:195          let network_size = self.
↳ crds.read().unwrap() .num_nodes();
   ./gossip/src/crds_gossip_push.rs:378     let mut ping_cache =
↳ ping_cache.lock().unwrap() ;
   ./ledger/src/blockstore.rs:407                    self.
↳ insert_shreds(shreds, None, false).unwrap() ;
   ./ledger/src/blockstore.rs:1370        shred.is_code() && shred.
↳ slot() > *last_root.read().unwrap()
   ./ledger/src/blockstore.rs:1706          let mut shredder =
↳ Shredder::new(current_slot, parent_slot, 0, version).unwrap() ;
   ./ledger/src/blockstore.rs:2568          let
↳ starting_primary_index = *self.active_transaction_status_index.
↳ read().unwrap() ;
   ./ledger/src/blockstore.rs:3243    *self.lowest_cleanup_slot.read
↳ ().unwrap()
   ./ledger/src/blockstore_processor.rs:860     ("slot", bank_forks.
↳ read().unwrap() .root(), i64),
   ./ledger/src/blockstore_processor.rs:862     ("forks", bank_forks.
↳ read().unwrap() .banks().len(), i64),
   ./ledger/src/blockstore_processor.rs:1397     if Some(bank_forks.
↳ read().unwrap() .root()) != opts.halt_at_slot {
   ./ledger/src/blockstore_processor.rs:1400       let (meta, bank,
↳ last_entry_hash) = pending_slots.pop().unwrap() ;
   ./ledger/src/shredder.rs:103          .unwrap()
   ./ledger/src/shredder.rs:116       .unwrap() ;
   ./ledger/src/shredder.rs:265         let shred = data.first()
↳ .unwrap() .borrow();
```

```
   ./ledger/src/shredder.rs:294          .unwrap()
   ./ledger/src/shredder.rs:296       .unwrap() ;
   ./ledger/src/shredder.rs:416       let mut cache = self.0.lock()
↳ .unwrap() ;
   ./ledger/src/shredder.rs:425         let mut cache = self.0.lock()
↳ .unwrap() ;
   ./ledger/src/shredder.rs:540     let next_index = data_shreds.last
↳ ().unwrap() .index() + 1;
   ./perf/src/perf_libs.rs:124        for entry in fs::read_dir(
↳ perf_libs_path).unwrap() .flatten() {
   ./rpc/src/rpc.rs:4532   blockstore.insert_shreds(shreds, None,
↳ false).unwrap() ;
   ./rpc/src/rpc.rs:4533   blockstore.set_roots(std::iter::once(&slot
↳ )).unwrap() ;
   ./runtime/src/accounts_db.rs:2196        .unwrap()
   ./runtime/src/accounts_db.rs:2215
↳ pubkeys_removed_from_accounts_index.into_inner().unwrap() ;
   ./runtime/src/bank/address_lookup_table.rs:36             .unwrap()
   ./runtime/src/snapshot_utils.rs:1570     let open_file = || File::
↳ open(snapshot_tar).unwrap() ;
```

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 any affected module, program or in the worst case the runtime 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.

MANUAL TESTING

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

4.1 GOSSIP

Description:

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. The service broadcasts information using a gossip protocol. Commit 2457c71e3ccf9c9c9c9e7a785a07b7aabe6651d7 backported to v1.14 a feature to ping peers before sending push message. A node sends a push message to tell the cluster it provides information to share. Upon receiving a push message, a node examines the message for: message duplication, new messages and expired messages.

The following requirements were tested:

- Sending pings before push messages does not modify the push message behavior.
- Ping timeouts do not introduce Denial of Service conditions.

Results:

No vulnerabilities were identified.

4.2 CONSENSUS

Description:

Commit 5571aca49f4465bab4a6768e282aeb3be767f64b manually backported rewrites removal, to maintain consensus and accounts hash calculation while developing and testing, and presumably during roll out, there was a system in place to allow any validator to skip rewrites and still maintain consensus. Several tests were performed to ensure that validators cannot skip rewrites.

Results:

No code vulnerabilities were identified.

4.3 BUSINESS PROCESS DESIGN

Description:

ExtendProgram is a BPF Upgradeable Loader instruction, which is permission less and allows extending the program data by a specified number of bytes, removing the need to pre-allocate program data accounts. Commit 565aacc23a4f9bfcc258e3ae5e75b911d2bbc9f8 modified the ExtendProgram instruction to ensure that not only the program data account is writable, but also the program account is. The following tests were performed:

- It is not possible to pass programdata_account and program_account as read accounts.
- Account ownership is validated.
- It is not possible to extend accounts when providing malformed or unintended accounts.
- Immutable programs cannot be extended.

Results:

No code vulnerabilities were identified.

4.4 ADDRESS LOOKUP TABLE

Description:

Address Lookup Table's (ALTs) allow developers to create a collection of addresses to load more addresses per in a single transaction, traditionally developers were limited to including 32 addresses per transaction but with ALT's it can be extended to 256. In commits 41eb7e and c32706 ALT's support was added to the CLI for and the account decoder. Several tests were done to check how the CLI, account decoder and ALTs program handles unexpected behavior and malformed data/accounts.

Results:

No code vulnerabilities were identified.

4.5 LOAD ZERO LAMPORTS

Description:

The Accounts and Accounts_db module has been updated in commit, when the Accounts::Load() function is called an account that has no lamports it will now return None. Several other functions also require a LoadZeroLamports struct to be supplied. Due to the criticality of these modules, they were tested to ensure they behave as expected.

Results:

No code vulnerabilities were identified.

AUTOMATED TESTING

5.1 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-2021-0139	ansi_term	ansi_term is unmaintained
RUSTSEC-2020-0016	net2	net2 crate has been deprecated
RUSTSEC-2021-0127	serde_cbor	serde_cbor is unmaintained
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘

5.2 UNSAFE RUST CODE DETECTION

Description:

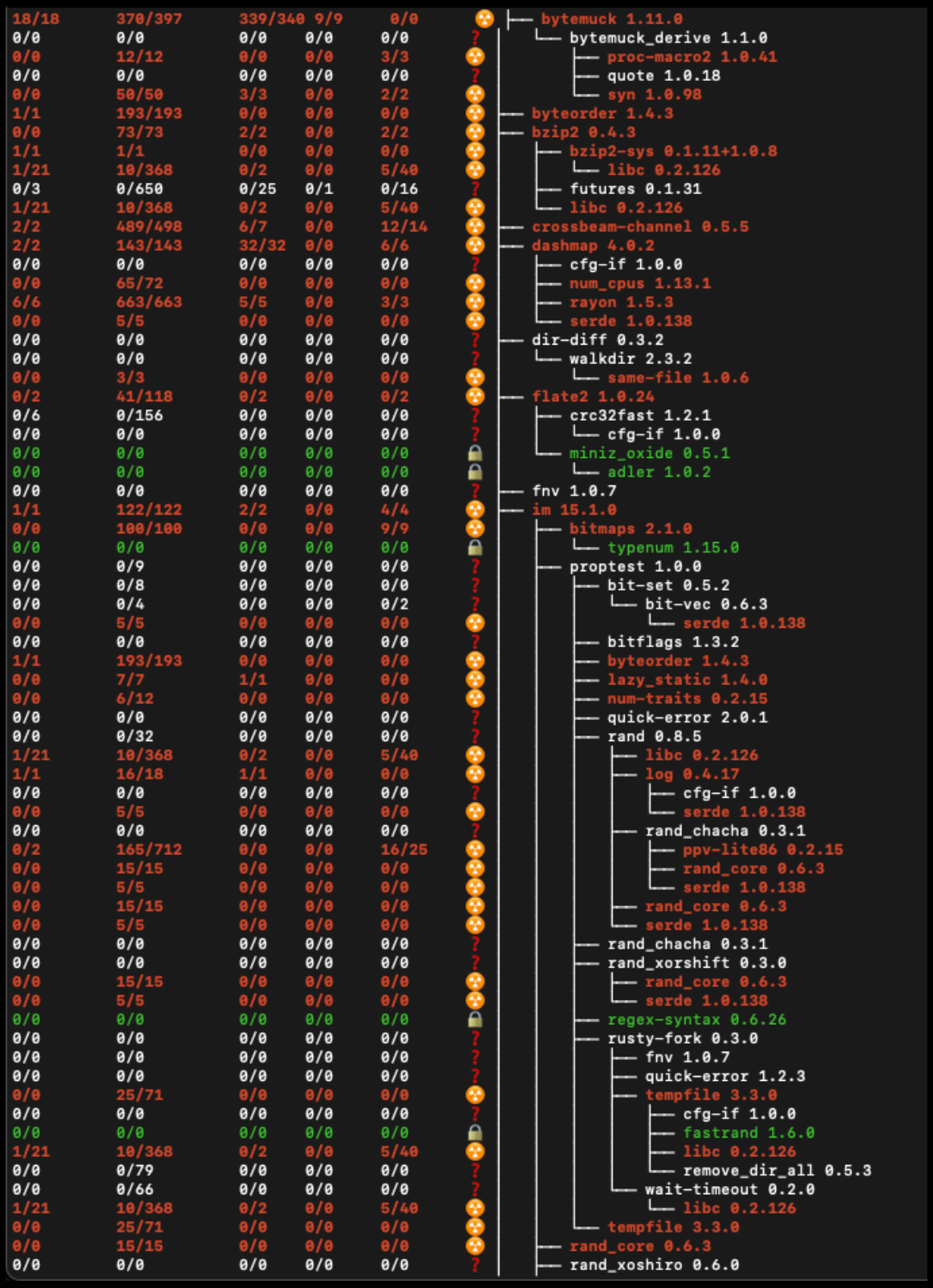
Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

AUTOMATED TESTING

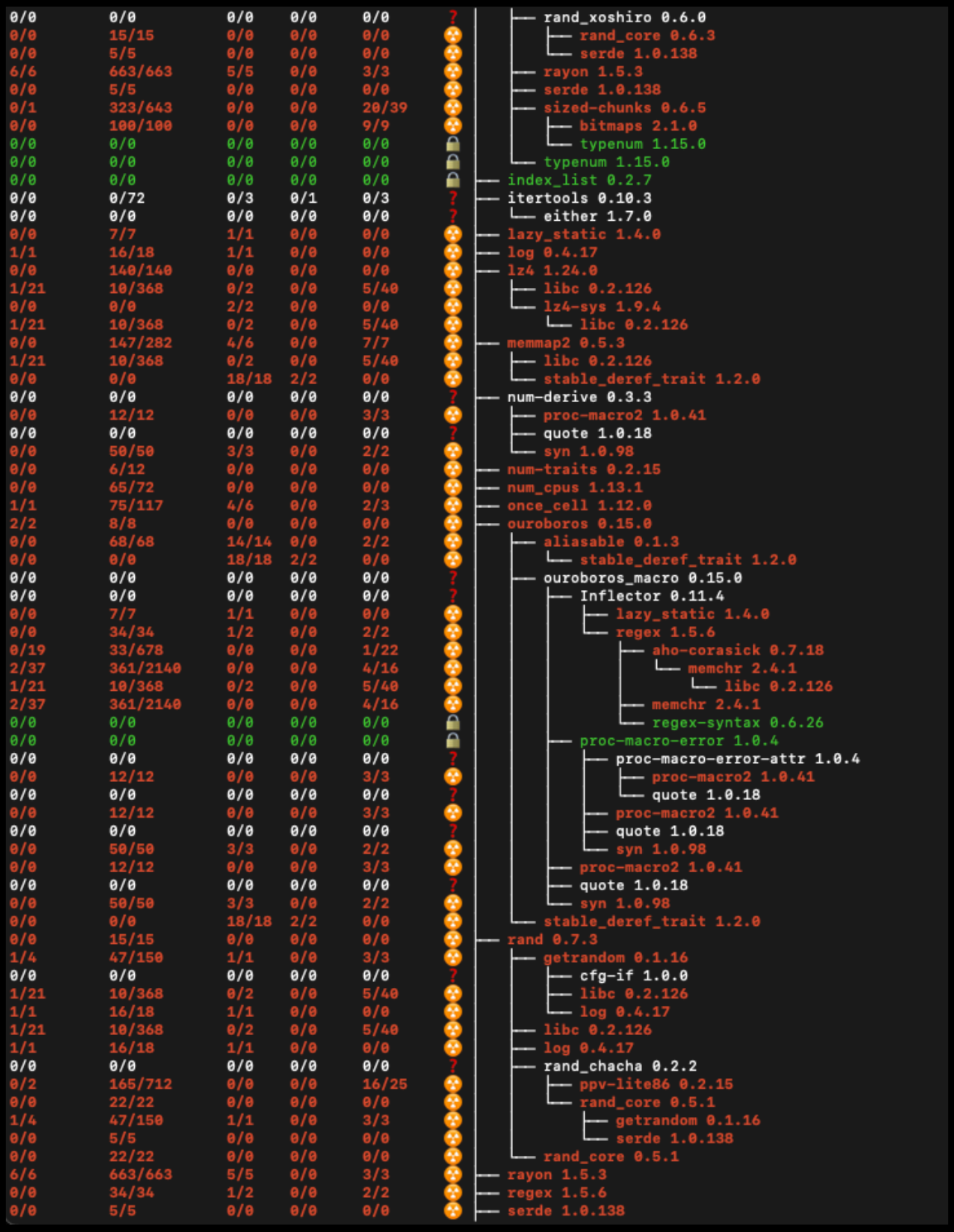
Results:

```
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        35/35        0/0    0/0     0/0      ☢ solana-runtime 1.14.6
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 1.0.138
0/0        0/0          0/0    0/0     0/0      ❓ │       └── serde_derive 1.0.138
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 1.0.138
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        292/292      20/20  8/8     5/5      ☢ │   │   │   └── generic-array 0.14.5
0/0        5/5          0/0    0/0     0/0      ☢ │   │   │       ├── serde 1.0.138
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │       └── typenum 1.15.0
0/0        0/0          0/0    0/0     0/0      🔒 │   │   ├── crypto-common 0.1.3
1/1        292/292      20/20  8/8     5/5      ☢ │   │   │   ├── generic-array 0.14.5
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 0.2.126
0/0        5/5          0/0    0/0     0/0      ☢ │   │   │   │   └── serde 1.0.138
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        0/0          0/0    0/0     0/0      ❓ │       ├── either 1.7.0
0/0        5/5          0/0    0/0     0/0      ☢ │       │   └── serde 1.0.138
5/5        485/488      2/2    0/0     20/20    ☢ │       └── rayon-core 1.9.2
2/2        489/498      6/7    0/0     12/14    ☢ │           ├── crossbeam-channel 0.5.5
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        451/451      6/6    0/0     6/6      ☢ │           ├── crossbeam-deque 0.8.1
4/4        85/85        14/14  0/0     2/2      ☢ │           ├── crossbeam-utils 0.8.8
0/0        65/72        0/0    0/0     0/0      ☢ │           └── num_cpus 1.13.1
1/21       10/368       0/2    0/0     5/40     ☢ │               └── libc 0.2.126
2/2        206/206      0/0    0/0     7/7      ☢ ├── bv 0.11.1
0/0        5/5          0/0    0/0     0/0      ☢ │   └── serde 1.0.138
18/18      370/397      339/340 9/9     0/0      ☢ ├── bytemuck 1.11.0
```

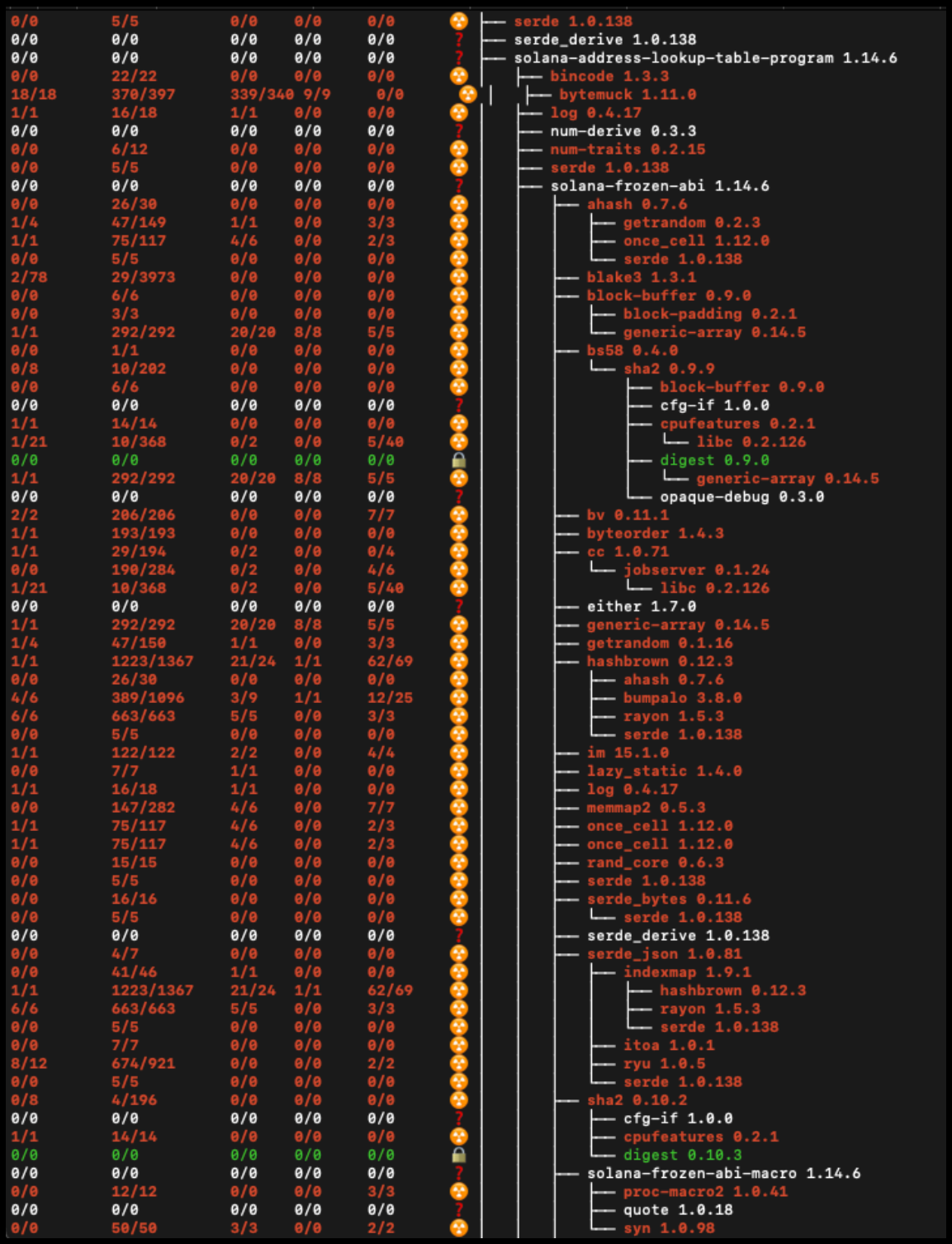
AUTOMATED TESTING



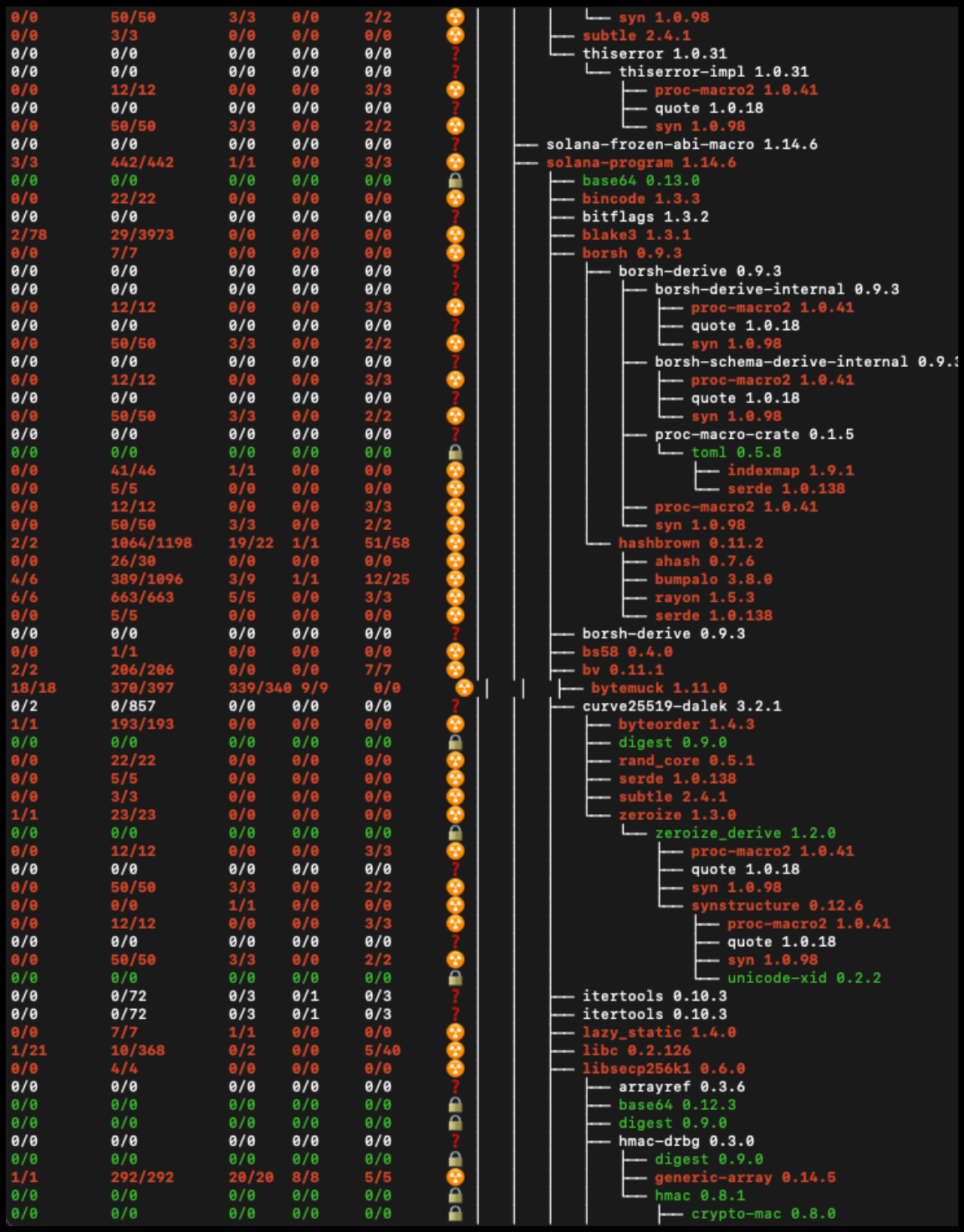
AUTOMATED TESTING



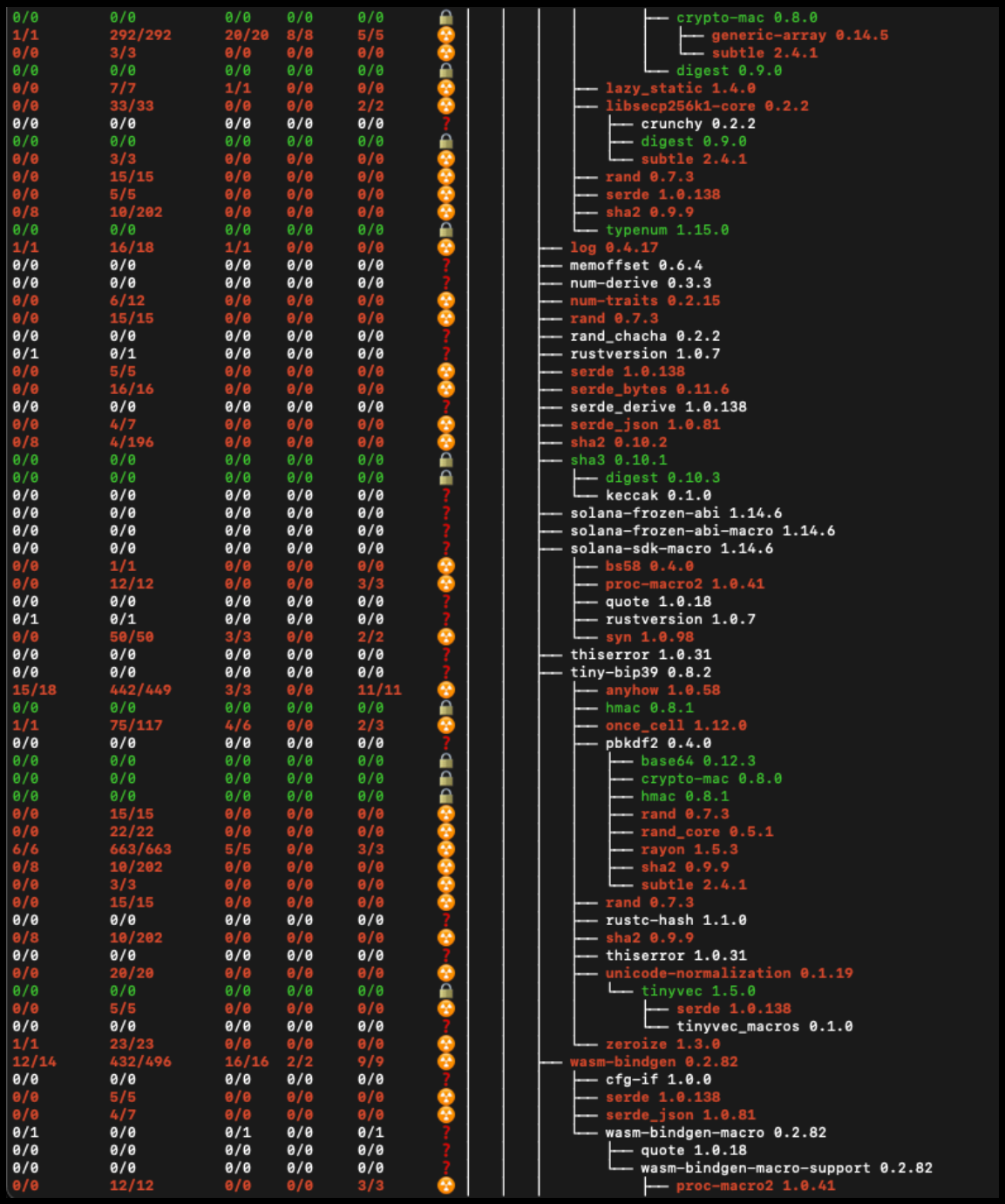
AUTOMATED TESTING



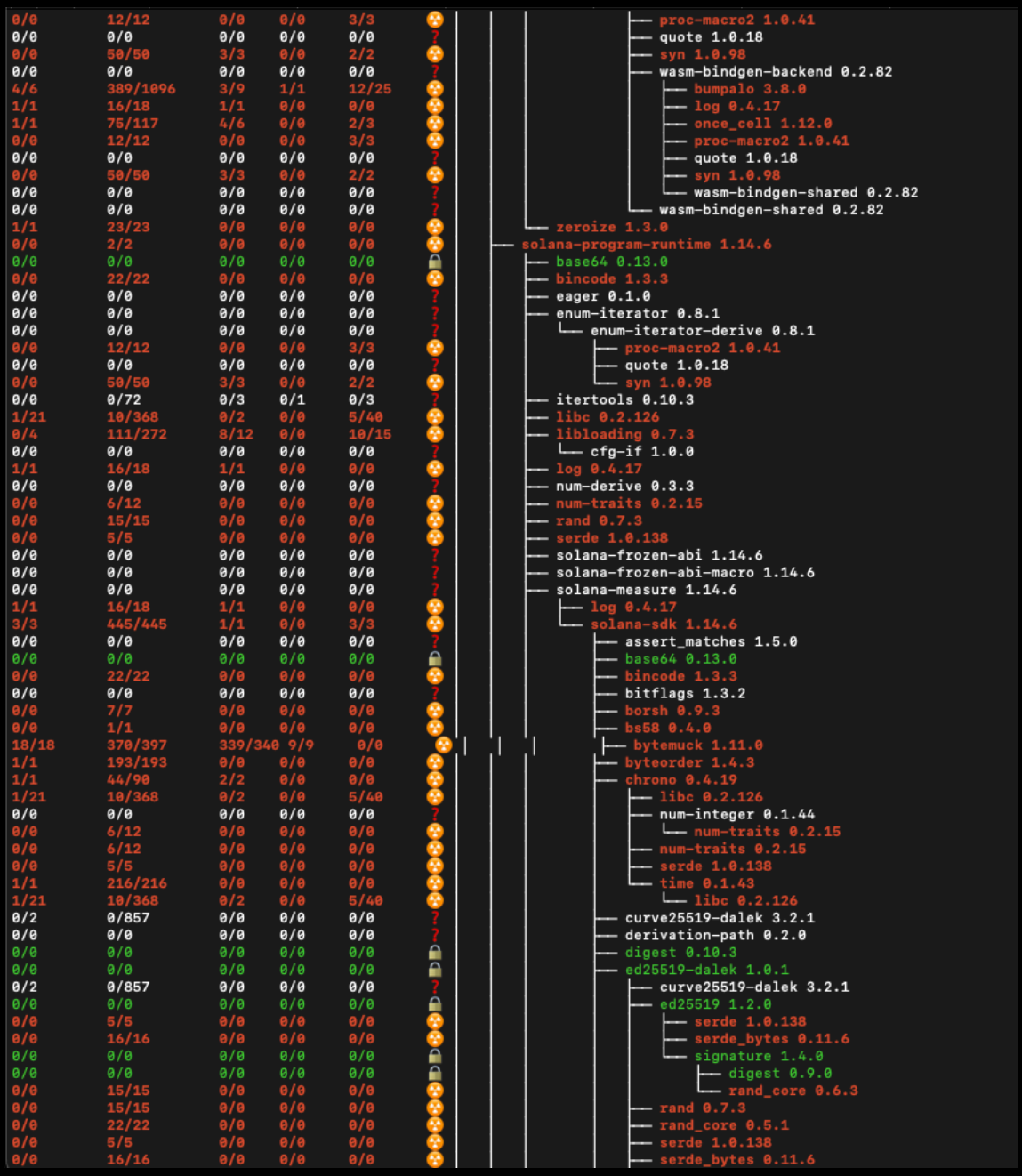
AUTOMATED TESTING



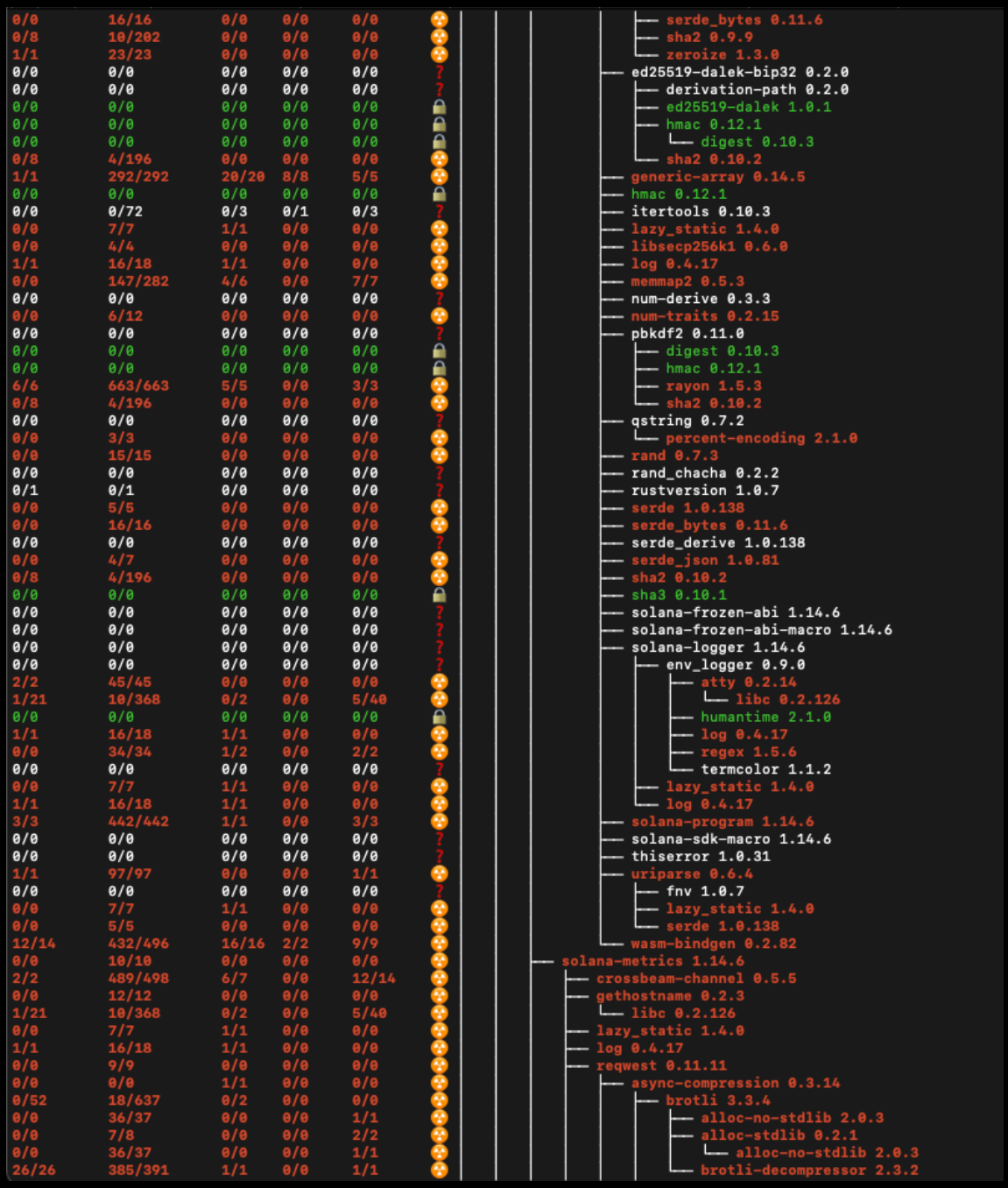
AUTOMATED TESTING



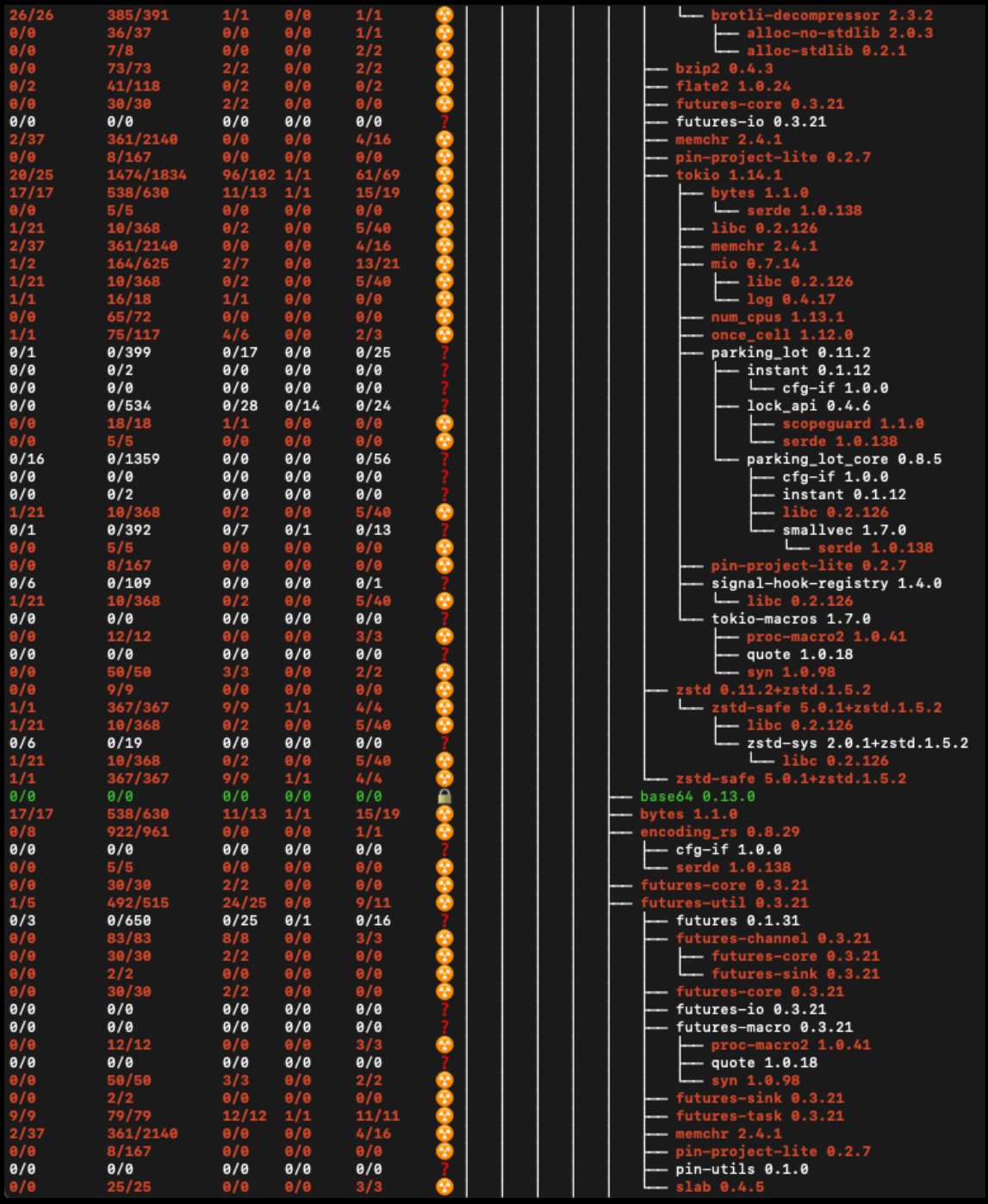
AUTOMATED TESTING



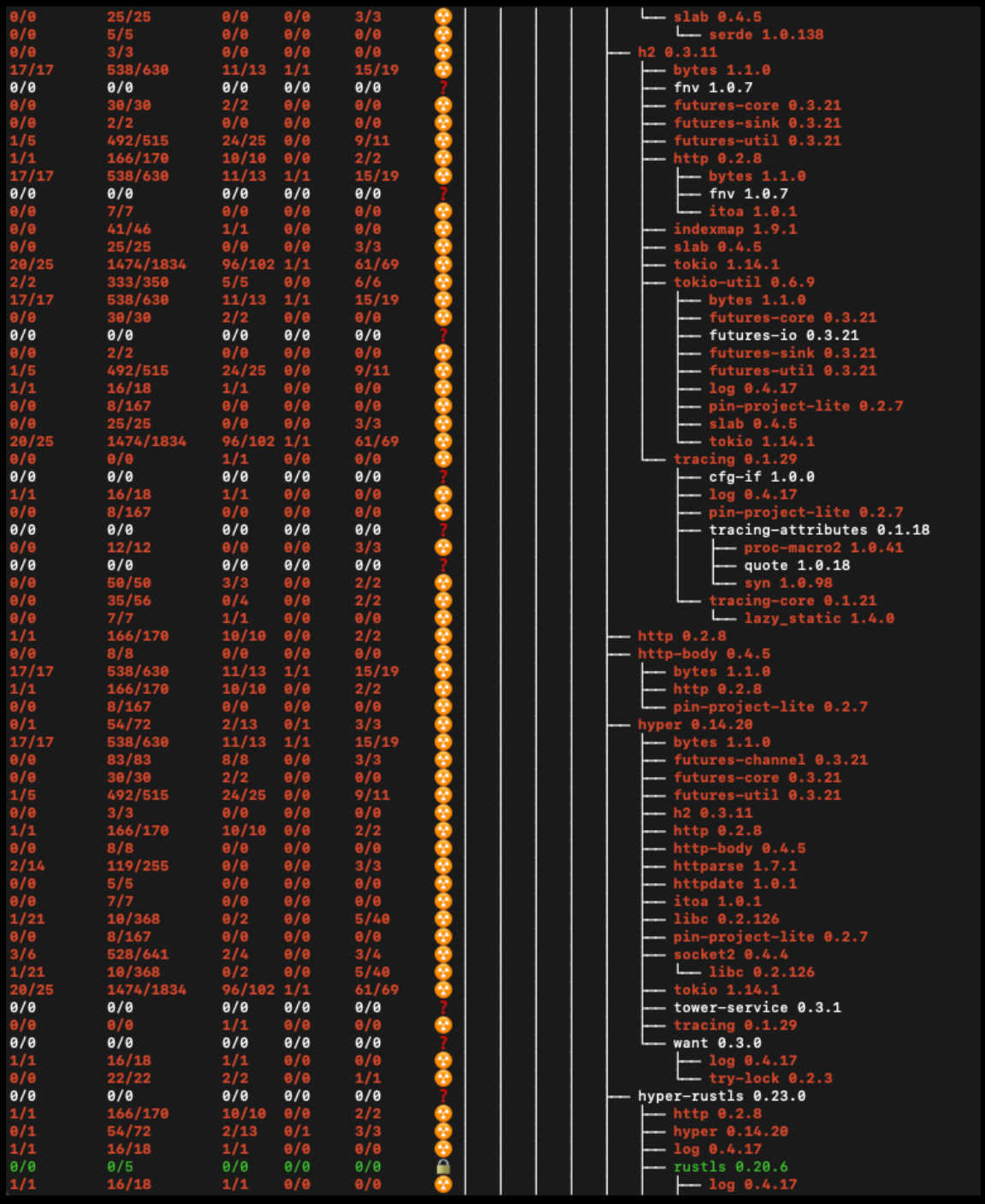
AUTOMATED TESTING



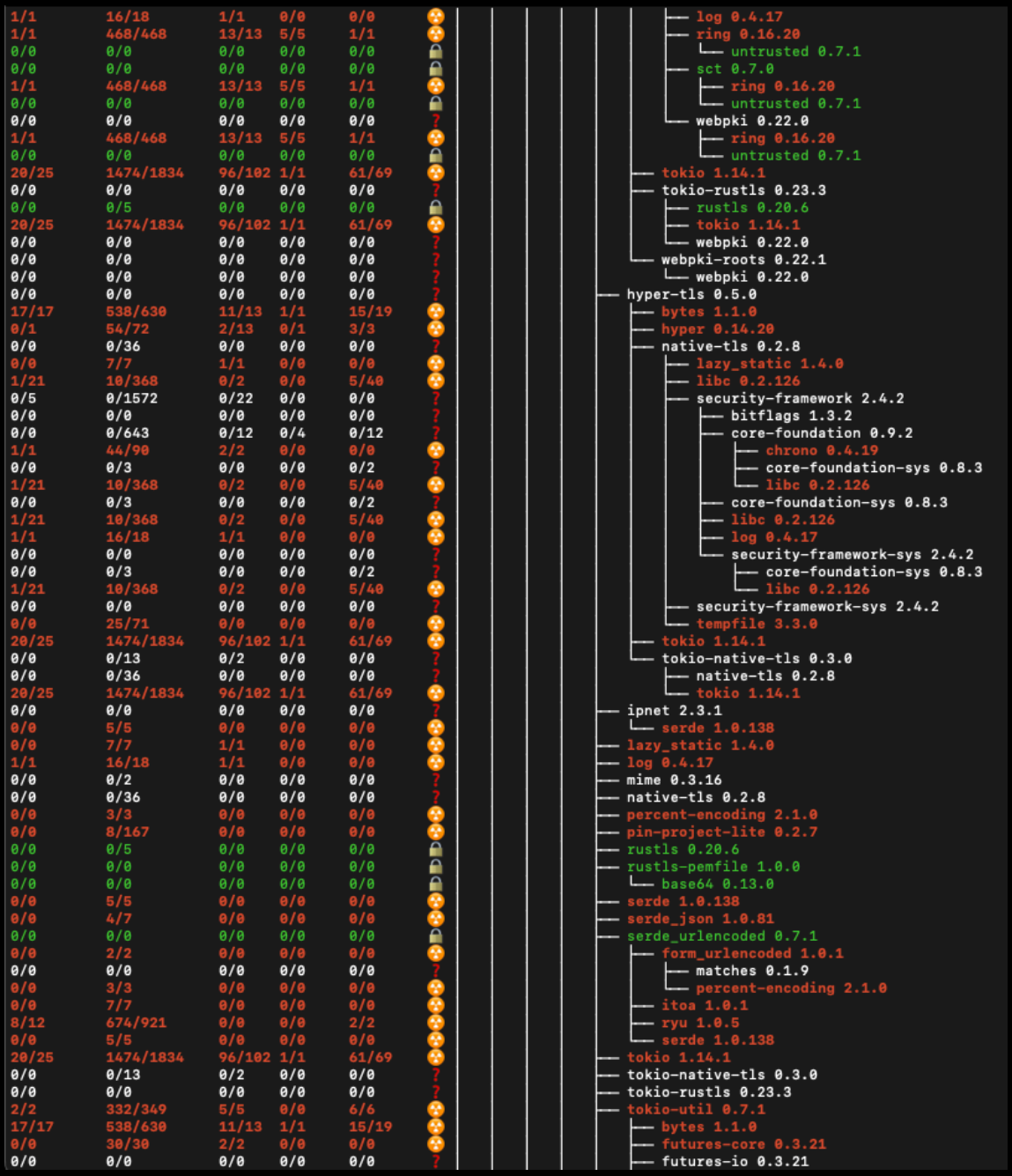
AUTOMATED TESTING



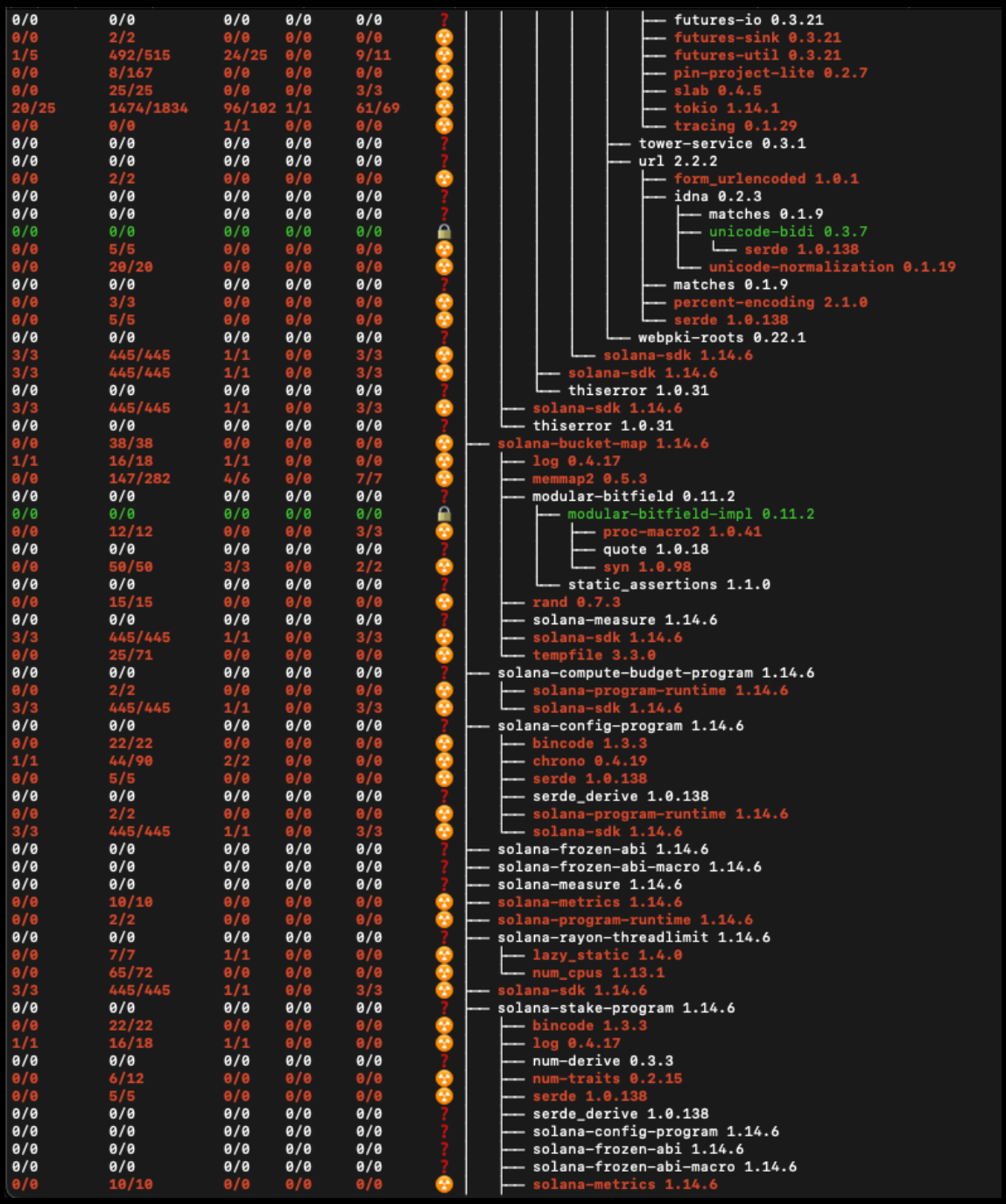
AUTOMATED TESTING



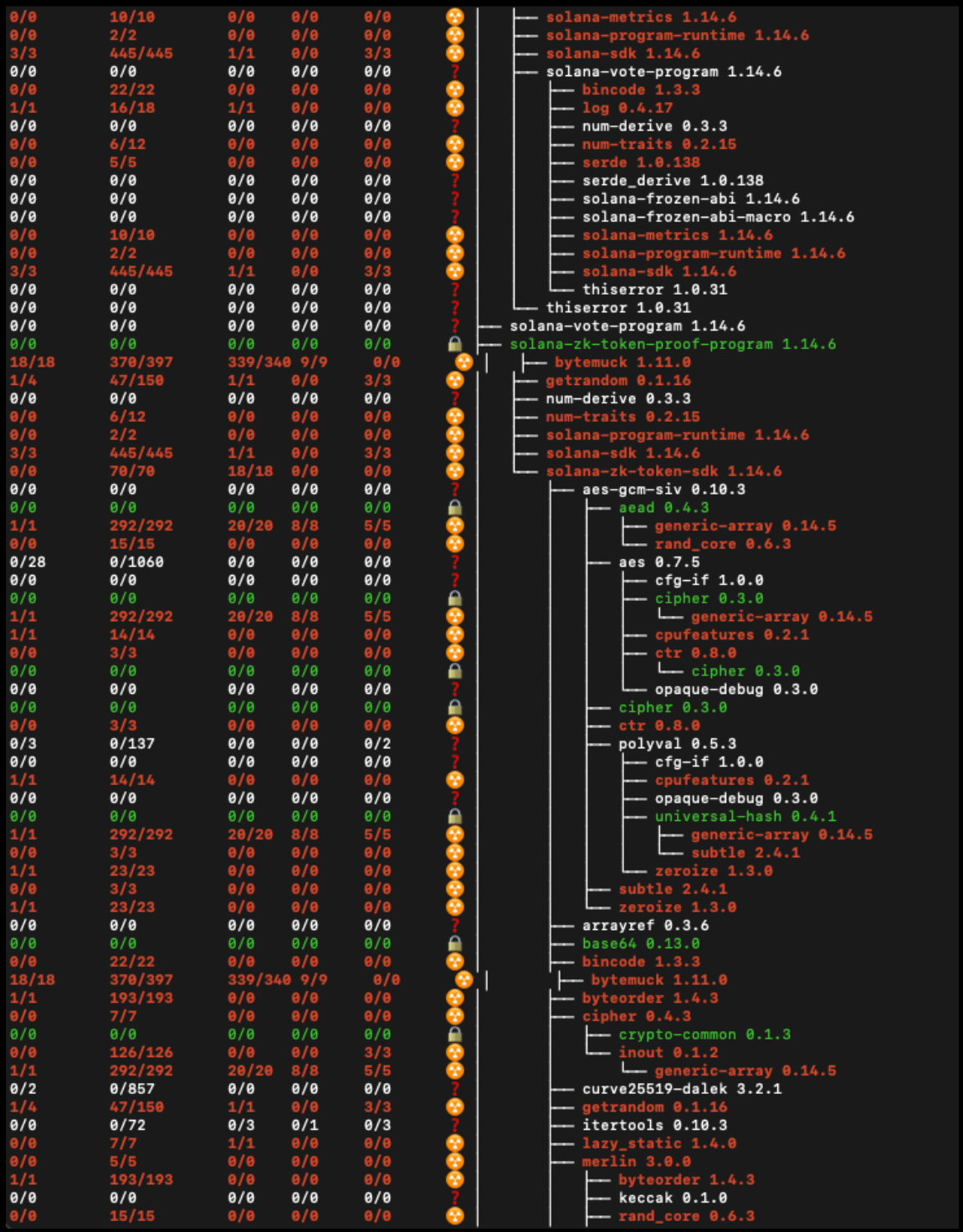
AUTOMATED TESTING



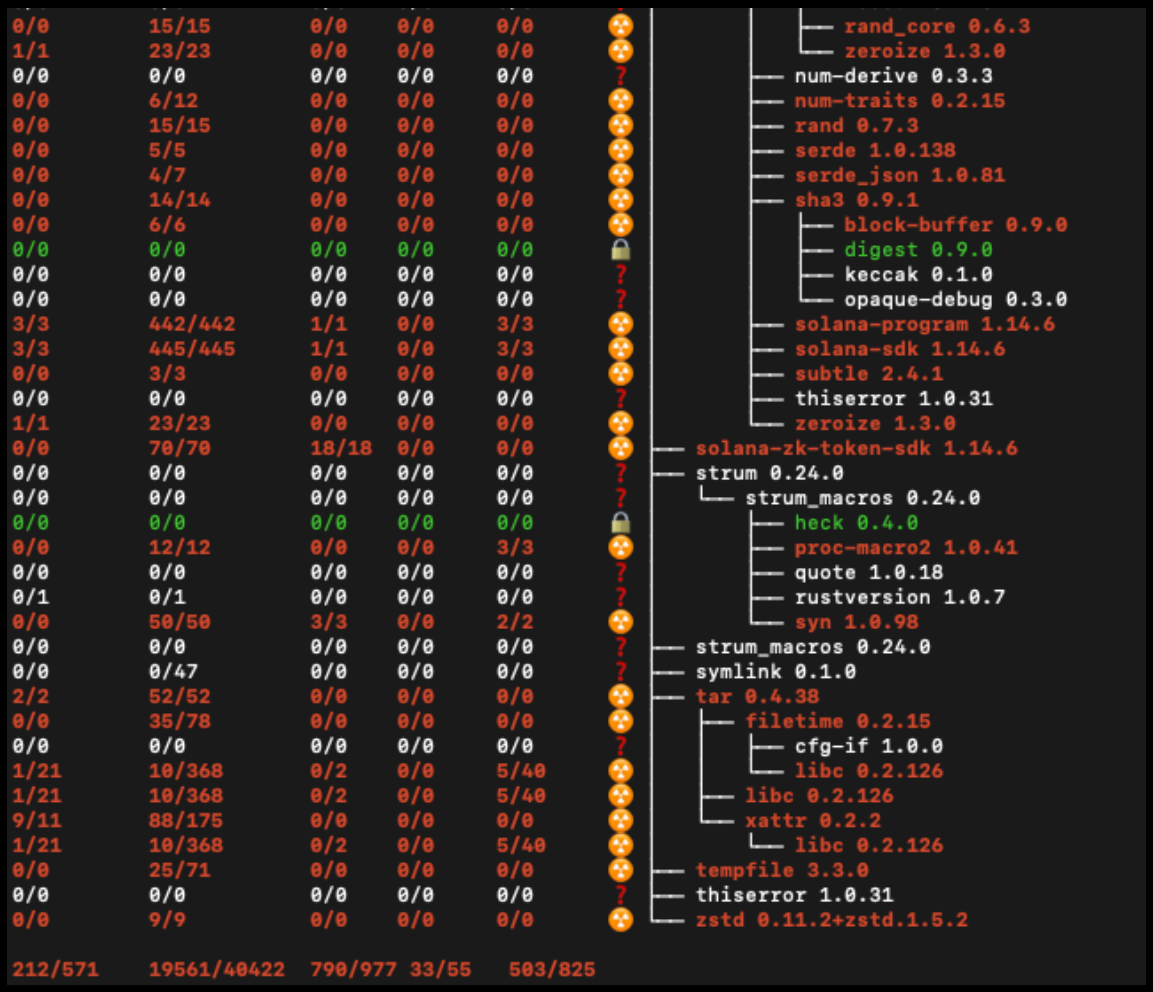
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN