HALBORN

Solana Labs - Runtime v1.14.13 - v1.14

L1 Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 24th, 2023 - May 5th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/27/2023	Isabel Burruezo
0.2	Document Updates	05/04/2023	Isabel Burruezo
0.3	Final Draft	05/04/2023	Isabel Burruezo
0.4	Draft Review	05/06/2023	Piotr Cielas
0.5	Draft Review	05/08/2023	Gabi Urrutia
1.0	Remediation Plan	05/18/2023	Isabel Burruezo
1.1	Remediation Plan Review	05/19/2023	Piotr Cielas
1.2	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

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 audit on the Solana v1.14.13 to v1.14 changes beginning on April 24th, 2023 and ending on May 5th, 2023 . The security assessment was scoped to the implementation of the updates up to v1.14 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 N weeks for the engagement and assigned M full-time security engineer/engineers to audit the security of the programs in scope. The security engineer/engineers is/are (a) blockchain and smart contract security expert/experts 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 in releases 1.14.13 to 1.14 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 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 led 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
- Releases v1.14.13-v1.14:
  - start: 0a3e52ba8b4f63c3675fa91fc89c4f54f69e5855
  - final: b93eafc44d0ea9c244934d1e4c703c8f2cea66e8
- 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
MISSING CARGO OVERFLOW CHECKS	Informational (0.0)	ACKNOWLEDGED
OUTDATED VERSION OF DEPENDENCIES	Informational (0.0)	SOLVED - 05/15/2023

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 finding.

4.2 (HAL-02) OUTDATED VERSION OF DEPENDENCIES - INFORMATIONAL (0.0)

Description:

The **remove_dir_all** crate is a Rust library that offers additional features over the Rust standard library **fs::remove_dir_all** function.

However, a vulnerability called Race Condition Enabling Link Following and Time-of-check Time-of-use, has recently been reported in those versions below 0.8.0. It affects the functions **remove_dir_all::ensure_empty_dir**, **remove_dir_all::remove_dir_all** and **remove_dir_all::remove_dir_contents**. This vulnerability makes it possible to trick a privileged process doing a recursive delete in an attacker controlled directory into deleting privileged files, on all operating systems.

It has been detected that the **remove_dir_all::remove_dir_all** function is used in some components, specifically accounts_db and snapshot_utils, which are within the scope of the scope.

Code Location:

Listing 1: runtime/src/accounts_db.rs (Line 6937)

```
6930 fn get_cache_hash_data(&self, config: &CalcAccountsHashConfig<'_>)
  ↳  -> CacheHashData {
6931         if !config.store_detailed_debug_info_on_failure {
6932             CacheHashData::new(&self.accounts_hash_cache_path)
6933         } else {
6934             // this path executes when we are failing with a hash
  ↳ mismatch
6935             let mut new = self.accounts_hash_cache_path.clone();
6936             new.push("failed_calculate_accounts_hash_cache");
6937             let _ = std::fs::remove_dir_all(&new);
6938             CacheHashData::new(&new)
6939         }
6940     }
```

Listing 2: runtime/src/accounts_db.rs (Line 6937)

```
6930
```

Listing 3

```
./local-cluster/tests/local_cluster_flakey.rs:238:        std::fs
↳ ::remove_dir_all(&validator_c_info.info.ledger_path).unwrap();
./local-cluster/tests/local_cluster_slow_2.rs:339:        std::fs
↳ ::remove_dir_all(&b_info.info.ledger_path).unwrap();
./ledger/src/blockstore.rs:4033:    let _ignored = fs::
↳ remove_dir_all(&path);
./ledger/src/bank_forks_utils.rs:97:        let _ = fs::
↳ remove_dir_all(&snapshot_config.bank_snapshots_dir);
./install/src/command.rs:166:        let _ = fs::remove_dir_all(
↳ extract_dir);
./install/src/command.rs:171:        let _ = fs::remove_dir_all(&
↳ tmp_extract_dir);
./install/src/command.rs:839:                let _ = fs::
↳ remove_dir_all(&release);
./install/src/command.rs:1177:    let _ = fs::remove_dir_all(
↳ config.active_release_dir());
./core/src/validator.rs:2092:    if let Err(e) = std::fs::
↳ remove_dir_all(account_path) {
./core/src/snapshot_packager_service.rs:248:        let _ignored =
↳  std::fs::remove_dir_all(&path);
./validator/src/bin/solana-test-validator.rs:843:            fs::
↳ remove_dir_all(&entry.path())?
./download-utils/src/lib.rs:239:        let _ignored = fs::
↳ remove_dir_all(&tmp_genesis_path);
./runtime/src/snapshot_utils.rs:276:                fs::
↳ remove_dir_all(entry.path())
./runtime/src/snapshot_utils.rs:822:    fs::remove_dir_all(
↳ bank_snapshot_dir)?;
./cli/src/program.rs:2332:        let _ignored = std::fs::
↳ remove_dir_all(&path);
./cli/src/cli.rs:1739:        let _ignored = std::fs::
↳ remove_dir_all(&path);
./ledger-tool/src/main.rs:929:            if let Err(err) = std::
↳ fs::remove_dir_all(&non_primary_accounts_path) {
./sdk/cargo-build-sbf/tests/crates.rs:52:    fs::remove_dir_all(
↳ target).expect("Failed to remove target dir");
./sdk/cargo-build-sbf/tests/crates.rs:94:    fs::remove_dir_all("
↳ tmp_out").expect("Failed to remove tmp_out dir");
```

```
20 ./sdk/cargo-build-sbf/src/main.rs:537:        fs::remove_dir_all(
↳ target_path_parent).unwrap_or_else(|err| {
21 ./sdk/src/genesis_config.rs:314:        let _ignored = std::fs::
↳ remove_dir_all(&path);
22 ./bucket_map/tests/bucket_map.rs:45:        std::fs::
↳ remove_dir_all(tmpdir).unwrap();
23 ./bucket_map/src/bucket_map.rs:108:            let _ = fs::
↳ remove_dir_all(folder);
24 ./accounts-bench/src/main.rs:68:    if fs::remove_dir_all(path.
↳ clone()).is_err() {
```

BVSS:

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

Recommendation:

Despite the low probability and high complexity of attack, it is highly recommended to update the related dependencies to use a patched version (>= 0.8.0).

It is also advisable to update the version of other crates used as dependencies, such as:
* h2 to patched version >=0.3.17.
* openssl to patched version >=0.10.48.

For more information, open the links in the table in the **AUTOMATED ANALYSIS** section.

Remediation Plan:

SOLVED: The Solana Labs team solved this issue by updating the versions of the mentioned dependencies to the recommended ones in commits d4e0d15 and 18f3699.

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 ENABLE REQUEST HEAP FRAME

Description:

In commit e2b476e3f4c0fbc9660dfa1b6684b274ea278a6e,
a new feature was added to enable or disable the co-existence of compute_budget::request_heap_frame and
and compute_budget::set_compute_unit_price instructions in the same transaction depending on the runtime version. In case the feature is disabled, the behavior of version v1.13 is maintained, and in case it is enabled, the coexistence mentioned above is possible and in addition, prioritization fees are counted in the transaction fees. This new feature was reviewed to ensure that the instruction can indeed coexist and that the prioritization fees are added correctly so that no vulnerability has been added that would result in a possible bypass of the payment of the corresponding fees or, on the contrary, that they are not added and are exempt.

Results:

No code vulnerabilities were identified.

```
running 1 test
[+]ENABLE REQ HEAP FRAME..
[*]bank calculating fees...
RequestHeapFrame instruction
bytes for req heap size: 263168
SetComputeUnitPrice instruction
micro lamports for prioritization_fee: 5000000
no unable req heap frame
Error: bytes > MAX_HEAP_FRAME_BYTES
prioritization_fee_details: PrioritizationFeeDetails { fee: 0, priority: 0 }
signature fee: 10
write_lock_fee: 0
compute fee: 0
thread 'bank::tests::test_calculate_fee_with_request_heap_frame_flag_max_heap_frame' panicked at 'assertion failed: `(left == right)`
  left: `10`,
 right: `15`', runtime/src/bank.rs:19239:9
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test bank::tests::test_calculate_fee_with_request_heap_frame_flag_max_heap_frame ... FAILED
```

MANUAL TESTING

```
running 1 test
[+]ENABLE REQ HEAP FRAME..
[*]bank calculating fees...
compute_unit_limit: 1
RequestHeapFrame instruction
bytes for req heap size: 0
SetComputeUnitPrice instruction
micro lamports for prioritization_fee: 5000000
no unable req heap frame
prioritization_fee_details: PrioritizationFeeDetails { fee: 0, priority: 0 }
signature fee: 10
write_lock_fee: 0
compute fee: 0
total_fee: 10
thread 'bank::tests::test_calculate_fee_with_request_heap_frame_flag_zero' panicked at 'assertion failed: `(left == right)`
  left: `10`,
 right: `15`', runtime/src/bank.rs:19328:9
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
test bank::tests::test_calculate_fee_with_request_heap_frame_flag_zero ... FAILED
```

```
[+]ENABLE REQ HEAP FRAME..
[*]bank calculating fees...
compute_unit_limit: 1
RequestHeapFrame instruction
bytes for req heap size: 40960
SetComputeUnitPrice instruction
micro lamports for prioritization_fee: 5000000
default_units_per_instruction:: true
updated_compute_unit_limit:: Some(1)
num_non_compute_budget_instructions:: 1
self.compute_unit_limit:: 1
PrioritizationFeeType::ComputeUnitPrice
cu_price: 5000000
micro_lamport_fee: 5000000
prioritization_fee_details: PrioritizationFeeDetails { fee: 5, priority: 5000000 }
signature fee: 10
write_lock_fee: 0
compute fee: 0
total_fee: 15
[+]]NOT ENABLE REQ HEAP FRAME..
[*]bank calculating fees...
compute_unit_limit: 1
RequestHeapFrame instruction
bytes for req heap size: 40960
SetComputeUnitPrice instruction
micro lamports for prioritization_fee: 5000000
Error: !enable_request_heap_frame_ix
no unable req heap frame
prioritization_fee_details: PrioritizationFeeDetails { fee: 0, priority: 0 }
signature fee: 10
write_lock_fee: 0
compute fee: 0
total_fee: 10
thread 'bank::tests::test_calculate_fee_with_request_heap_frame_flag_max_heap_frame' panicked at 'assertion failed: `(left == right)`
  left: `10`,
 right: `15`', runtime/src/bank.rs:19269:9
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

MANUAL TESTING

```
running 1 test
[+]ENABLE REQ HEAP FRAME..
[*]bank calculating fees...
compute_unit_limit: 1
RequestHeapFrame instruction
bytes for req heap size: 40960
default_units_per_instruction:: true
updated_compute_unit_limit:: Some(1)
num_non_compute_budget_instructions:: 1
self.compute_unit_limit:: 1
prioritization_fee_details: PrioritizationFeeDetails { fee: 0, priority: 0 }
signature fee: 10
write_lock_fee: 0
compute fee: 0
total_fee: 10
[+]]NOT ENABLE REQ HEAP FRAME..
[*]bank calculating fees...
compute_unit_limit: 1
RequestHeapFrame instruction
bytes for req heap size: 40960
Error: !enable_request_heap_frame_ix
no unable req heap frame
prioritization_fee_details: PrioritizationFeeDetails { fee: 0, priority: 0 }
signature fee: 10
write_lock_fee: 0
compute fee: 0
total_fee: 10
```

5.2 CURVE25519 SYSCALL MULTI-SCALAR MULTIPLICATION

Description:

Commit 486457db0f3241747fb1388a3bfa0bdf9831fc60 introduced a new syscall for multi-scalar multiplication. Multi-scalar multiplication (MSM) on elliptic curves is a core primitive of zero-knowledge proof systems, which are used to prove certain properties of encrypted data. In Solana, zero-knowledge proofs are used by the confidential token extension. In the merge request curve25519 Edwards and Ristretto representations were added leveraging the curve25519-dalek crate for the computation. Tests were performed to validate that given some points and scalars, the result was the expected. Additionally, it was checked that when passing invalid memory mappings to the syscall and error was successfully triggered.

Results:

No code vulnerabilities were identified.

```
running 1 test
N: 512
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 256
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 128
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 64
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 32
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 16
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 8
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 4
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 2
RESULT POINT: PodEdwardsPoint([193, 45, 141, 232, 30, 172, 115, 78, 80, 155, 233, 6, 85, 190, 54, 52, 249, 232, 68, 114, 226, 189, 234, 179, 21, 124, 222, 185, 32, 69, 41, 242])
N: 1
RESULT POINT: PodEdwardsPoint([58, 97, 203, 117, 25, 25, 108, 174, 65, 35, 22, 175, 181, 33, 98, 49, 58, 58, 67, 37, 97, 34, 81, 46, 5, 218, 51, 111, 179, 162, 184, 183])
test syscalls::tests::test_msm ... ok
```

5.3 PRIORITIZATION FEES CACHE

Description:

Priority fees allow users to have more control over when transactions are confirmed even at times when the network is congested, if the fee is high enough, the transaction should land in a block. The PrioritizationFeeCache keeps track of the associated fees in the last 150 blocks.

In commit 299a506822825e849c6a1e556628b334039f3f46, a new change was added to the method for updating **fee_cache** to ensure that transactions requesting zero **compute_unit_limit** are filtered out since their priority fees amount is not instructive. This change was reviewed to ensure the proper and expected functioning of this update and that no security risks have been accidentally introduced.

Results:

No code vulnerabilities were identified.

```
running 1 test
SetComputeUnitPrice instruction
micro lamports for prioritization_fee: 2
compute_unit_limit: 0
default_units_per_instruction:: true
updated_compute_unit_limit:: Some(0)
num_non_compute_budget_instructions:: 1
self.compute_unit_limit:: 0
PrioritizationFeeType::ComputeUnitPrice
cu_price: 2
micro_lamport_fee: 0
priorization_fee details: PrioritizationFeeDetails { fee: 0, priority: 2 }
priority_details: Some(TransactionPriorityDetails { priority: 2, compute_unit_limit: 0 })
priority_details.compute_unit_limit: 0
filtering out tx requesting zero compute unit limit
SetComputeUnitPrice instruction
micro lamports for prioritization_fee: 1
compute_unit_limit: 0
default_units_per_instruction:: true
updated_compute_unit_limit:: Some(0)
num_non_compute_budget_instructions:: 1
self.compute_unit_limit:: 0
PrioritizationFeeType::ComputeUnitPrice
cu_price: 1
micro_lamport_fee: 0
priorization_fee details: PrioritizationFeeDetails { fee: 0, priority: 1 }
priority_details: Some(TransactionPriorityDetails { priority: 1, compute_unit_limit: 0 })
priority_details.compute_unit_limit: 0
filtering out tx requesting zero compute unit limit
test prioritization_fee_cache::tests::test_get_prioritization_fees has been running for over 60 seconds
```

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-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations
RUSTSEC-2023-0034	h2	Resource exhaustion vulnerability in h2 may lead to Denial of Service (DoS)
RUSTSEC-2023-0023	openssl	‘openssl‘ ‘SubjectAlternativeName‘ and ‘ExtendedKeyUsage::other‘ allow arbitrary file read
RUSTSEC-2023-0022	openssl	‘openssl‘ ‘X509NameBuilder::build‘ returned object is not thread safe
RUSTSEC-2023-0024	openssl	‘openssl‘ ‘X509Extension::new‘ and ‘X509Extension::new_nid‘ null pointer dereference
RUSTSEC-2023-0018	remove_dir_all	Race Condition Enabling Link Following and Time-of-check Time-of-use (TOCTOU)

6.2 UNSAFE RUST CODE DETECTION

Description:

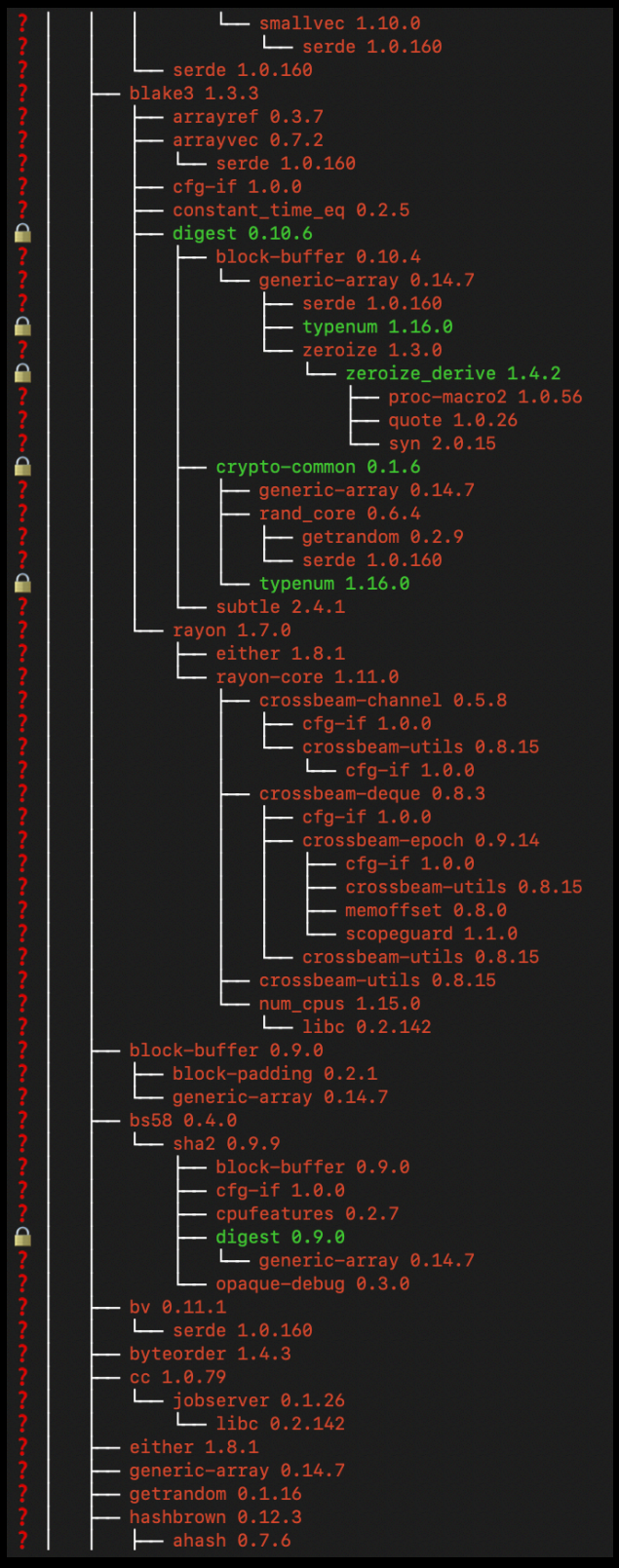
Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

Results:

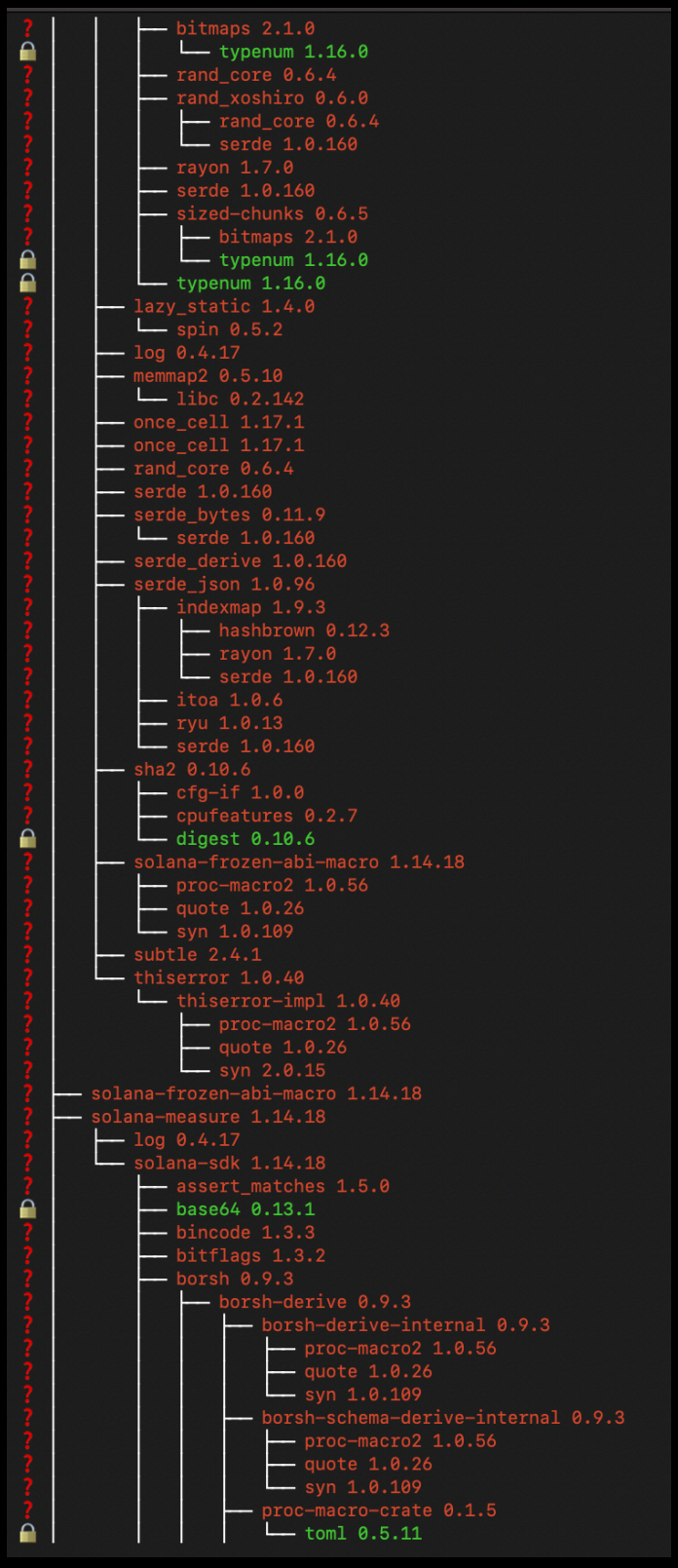
```
Symbols:
    🔒 = All entry point .rs files declare #![forbid(unsafe_code)].
    ❓ = This crate may use unsafe code.

❓ solana-program-runtime 1.14.18
🔒 ├── base64 0.13.1
❓ ├── bincode 1.3.3
❓ │   └── serde 1.0.160
❓ │       └── serde_derive 1.0.160
❓ │           ├── proc-macro2 1.0.56
❓ │           │   └── unicode-ident 1.0.8
❓ │           ├── quote 1.0.26
❓ │           │   └── proc-macro2 1.0.56
❓ │           └── syn 2.0.15
❓ │               ├── proc-macro2 1.0.56
❓ │               ├── quote 1.0.26
❓ │               └── unicode-ident 1.0.8
❓ ├── eager 0.1.0
❓ ├── enum-iterator 0.8.1
❓ │   └── enum-iterator-derive 0.8.1
❓ │       ├── proc-macro2 1.0.56
❓ │       ├── quote 1.0.26
❓ │       └── syn 1.0.109
❓ │           ├── proc-macro2 1.0.56
❓ │           ├── quote 1.0.26
❓ │           └── unicode-ident 1.0.8
❓ ├── itertools 0.10.5
❓ │   └── either 1.8.1
❓ │       └── serde 1.0.160
❓ ├── libc 0.2.142
❓ ├── libloading 0.7.4
❓ │   └── cfg-if 1.0.0
❓ ├── log 0.4.17
❓ │   ├── cfg-if 1.0.0
❓ │   └── serde 1.0.160
❓ ├── num-derive 0.3.3
❓ │   ├── proc-macro2 1.0.56
❓ │   ├── quote 1.0.26
❓ │   └── syn 1.0.109
❓ ├── num-traits 0.2.15
❓ │   └── libm 0.2.6
❓ ├── rand 0.7.3
❓ │   ├── getrandom 0.1.16
❓ │   │   ├── cfg-if 1.0.0
❓ │   │   ├── libc 0.2.142
❓ │   │   └── log 0.4.17
❓ │   ├── libc 0.2.142
❓ │   ├── log 0.4.17
❓ │   ├── rand_chacha 0.2.2
❓ │   │   ├── ppv-lite86 0.2.17
❓ │   │   └── rand_core 0.5.1
❓ │   │       ├── getrandom 0.1.16
❓ │   │       └── serde 1.0.160
❓ │   └── rand_core 0.5.1
❓ ├── serde 1.0.160
❓ ├── solana-frozen-abi 1.14.18
❓ │   ├── ahash 0.7.6
❓ │   │   ├── getrandom 0.2.9
❓ │   │   │   ├── cfg-if 1.0.0
❓ │   │   │   └── libc 0.2.142
❓ │   │   ├── once_cell 1.17.1
❓ │   │   │   └── parking_lot_core 0.9.7
❓ │   │   │       ├── cfg-if 1.0.0
❓ │   │   │       ├── libc 0.2.142
```

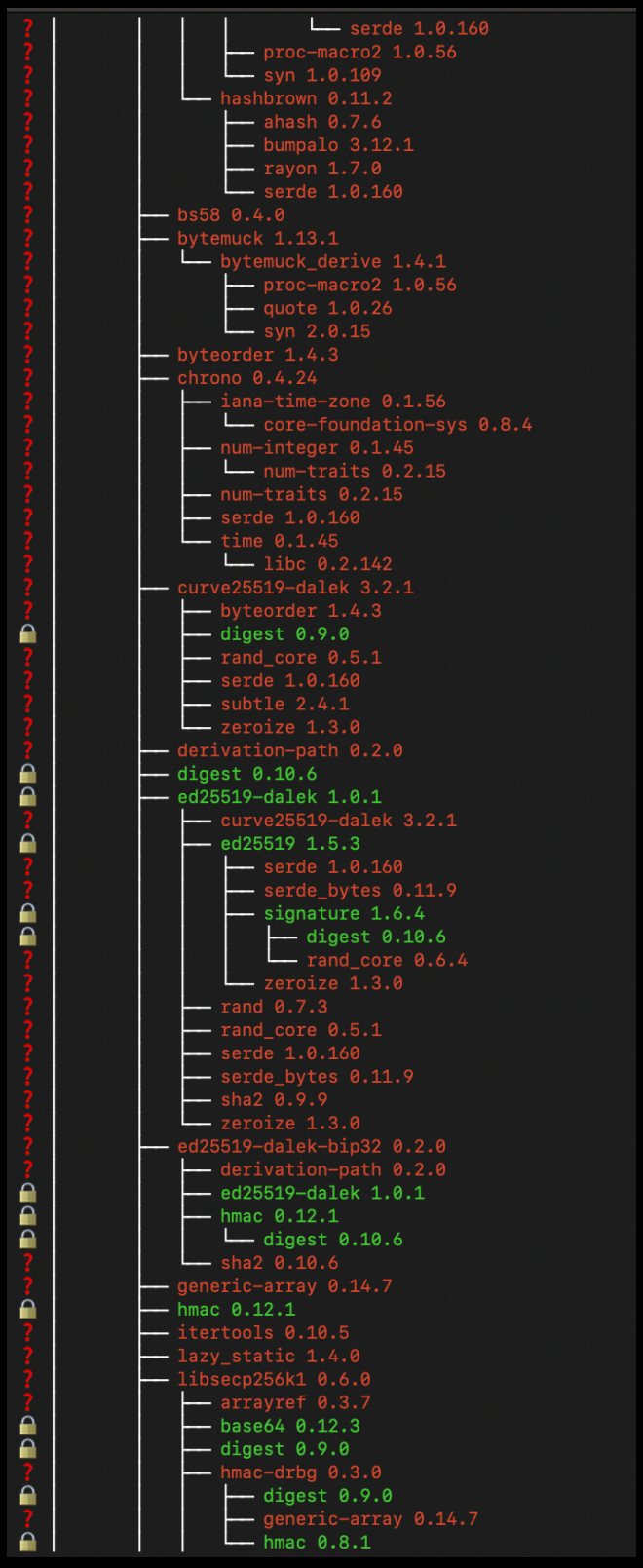
AUTOMATED TESTING



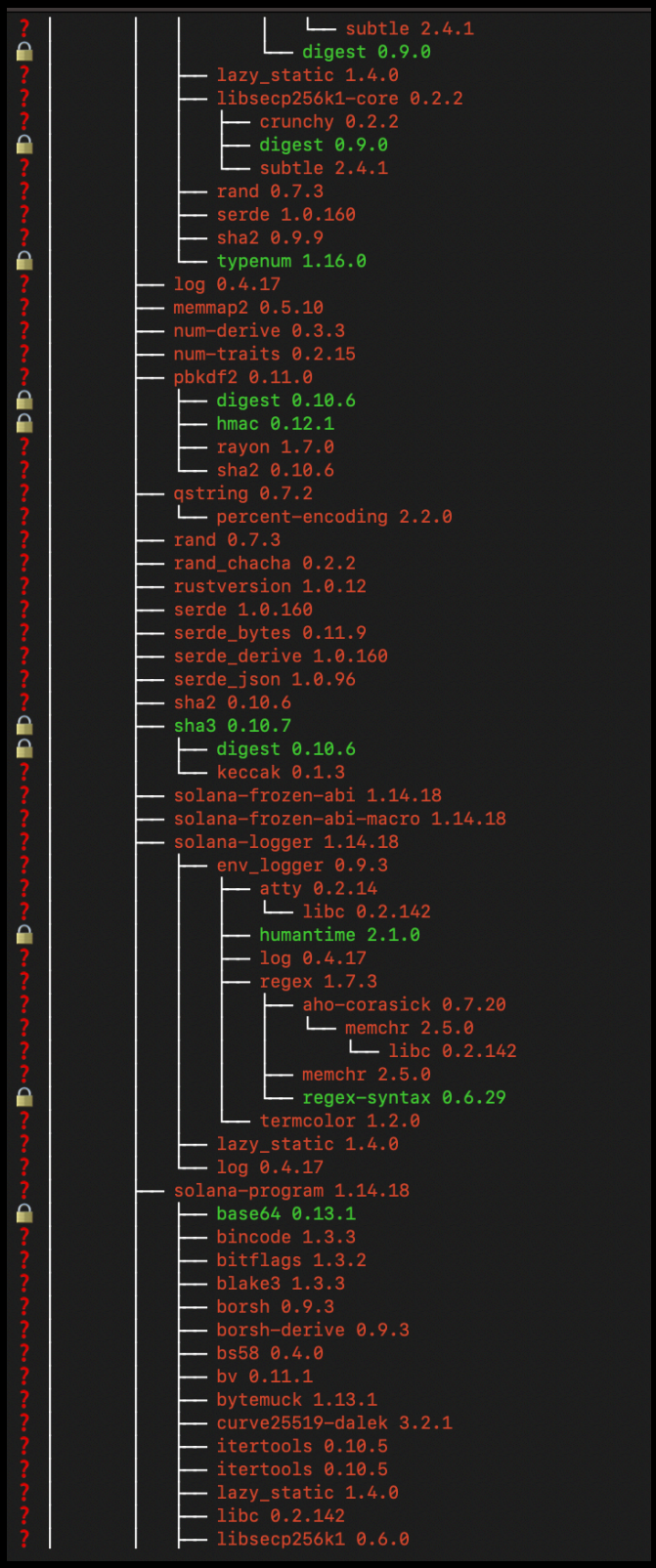
AUTOMATED TESTING



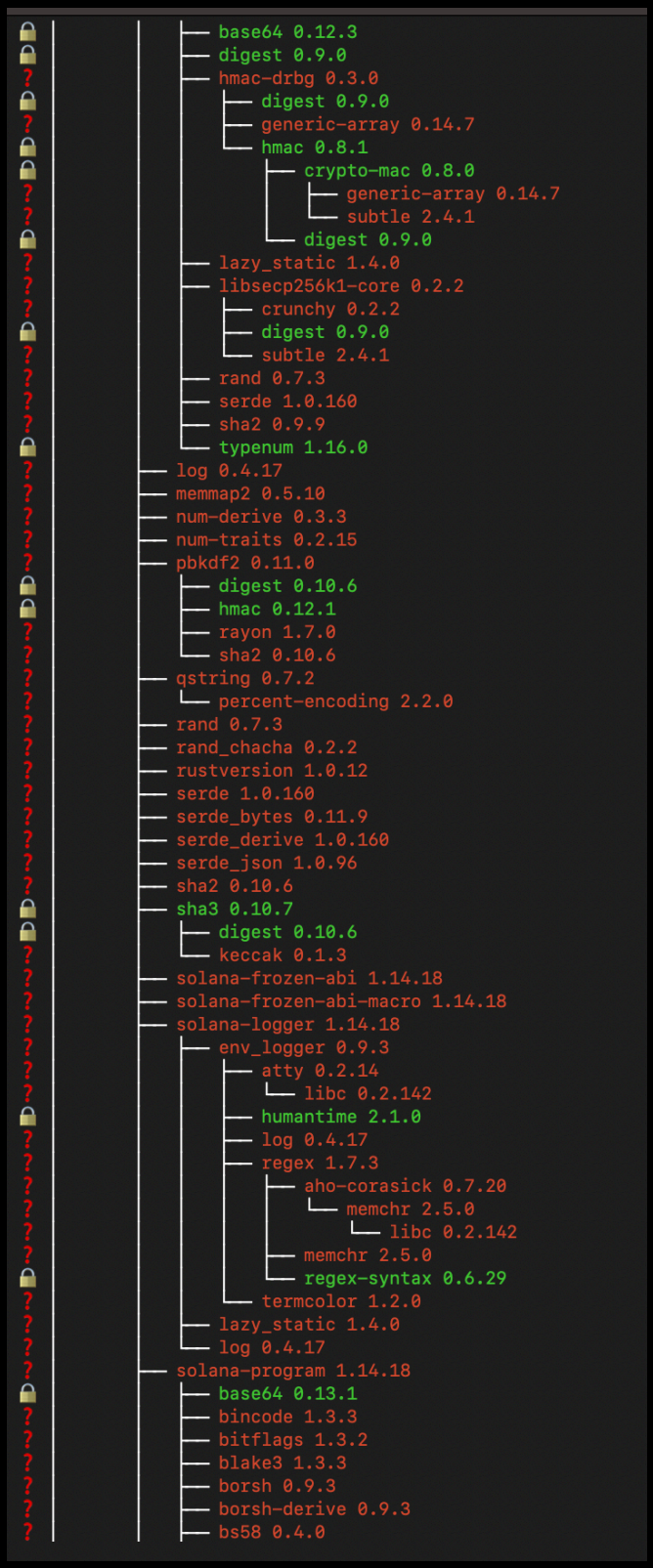
AUTOMATED TESTING



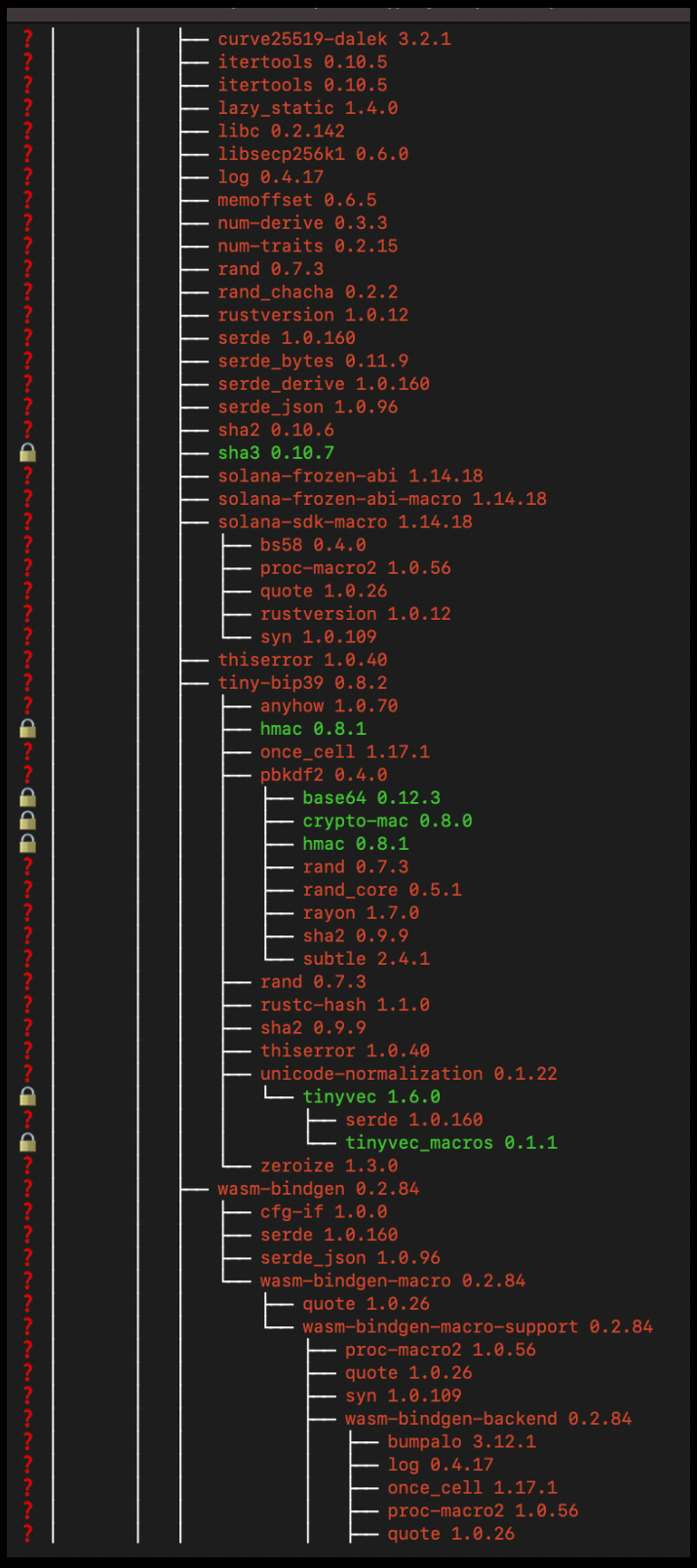
AUTOMATED TESTING



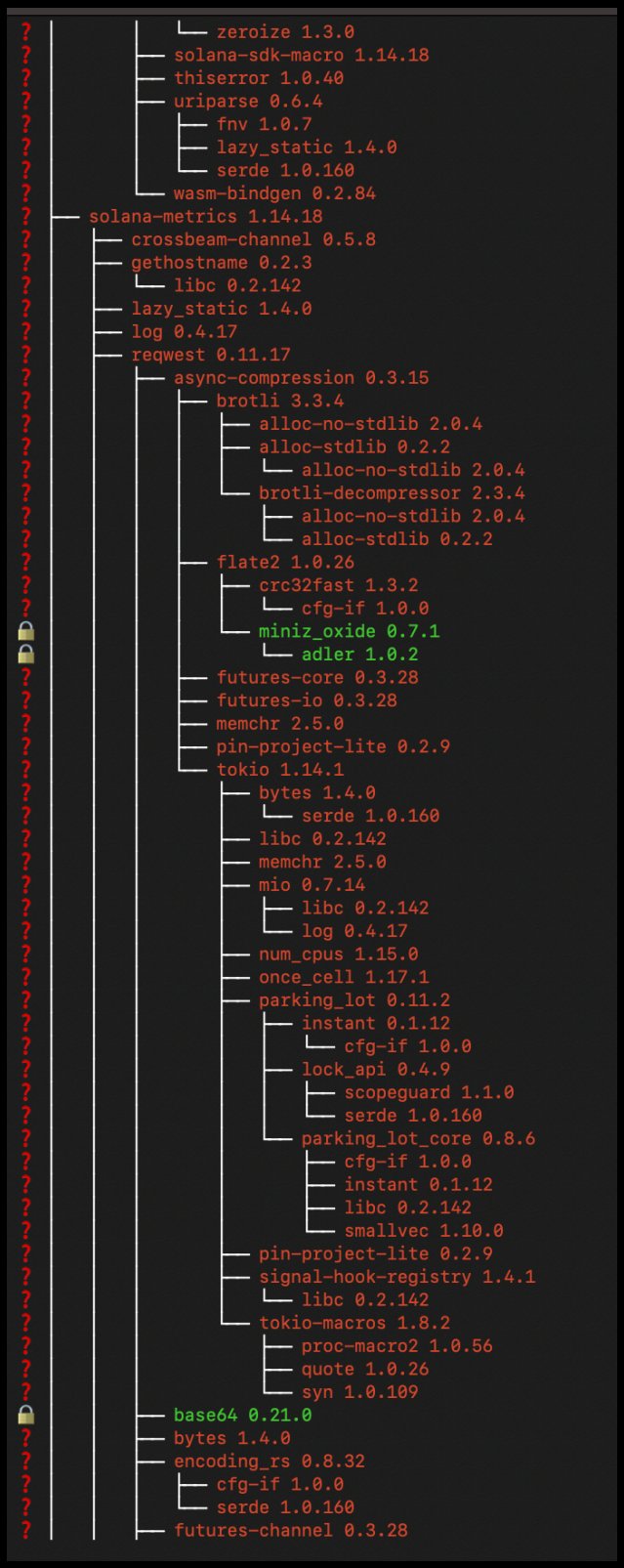
AUTOMATED TESTING



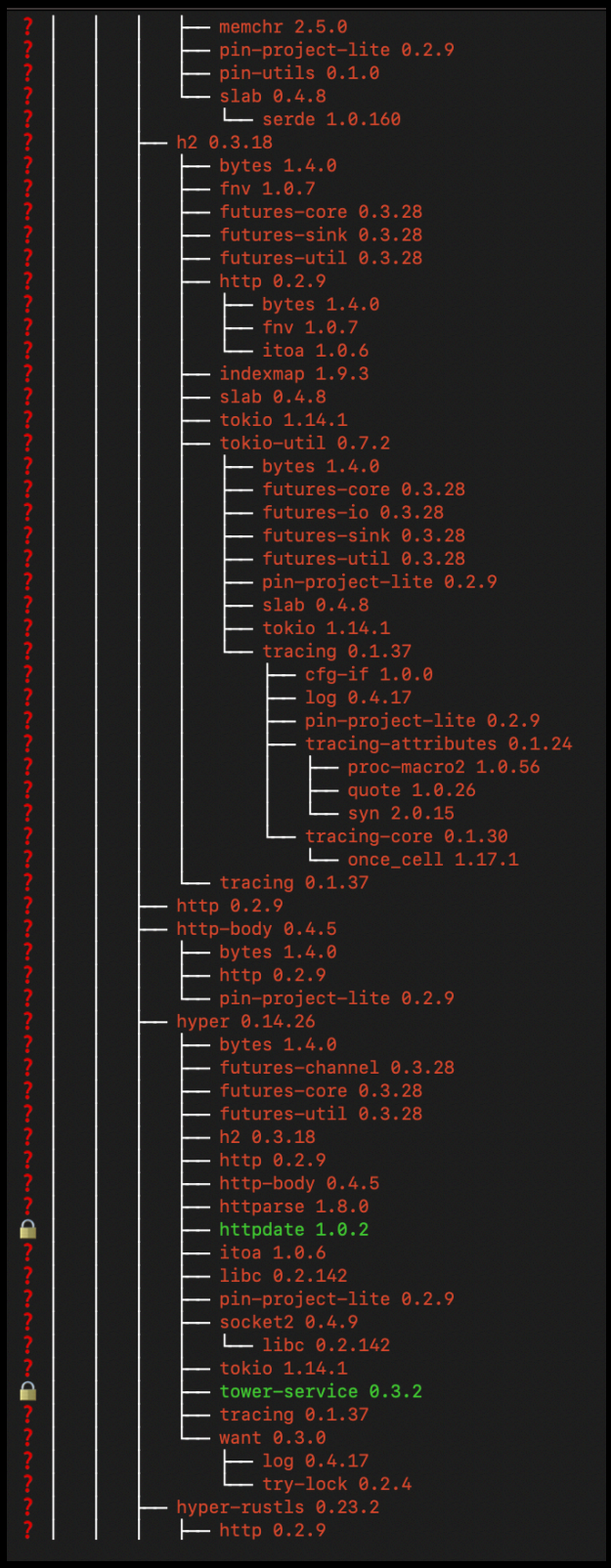
AUTOMATED TESTING



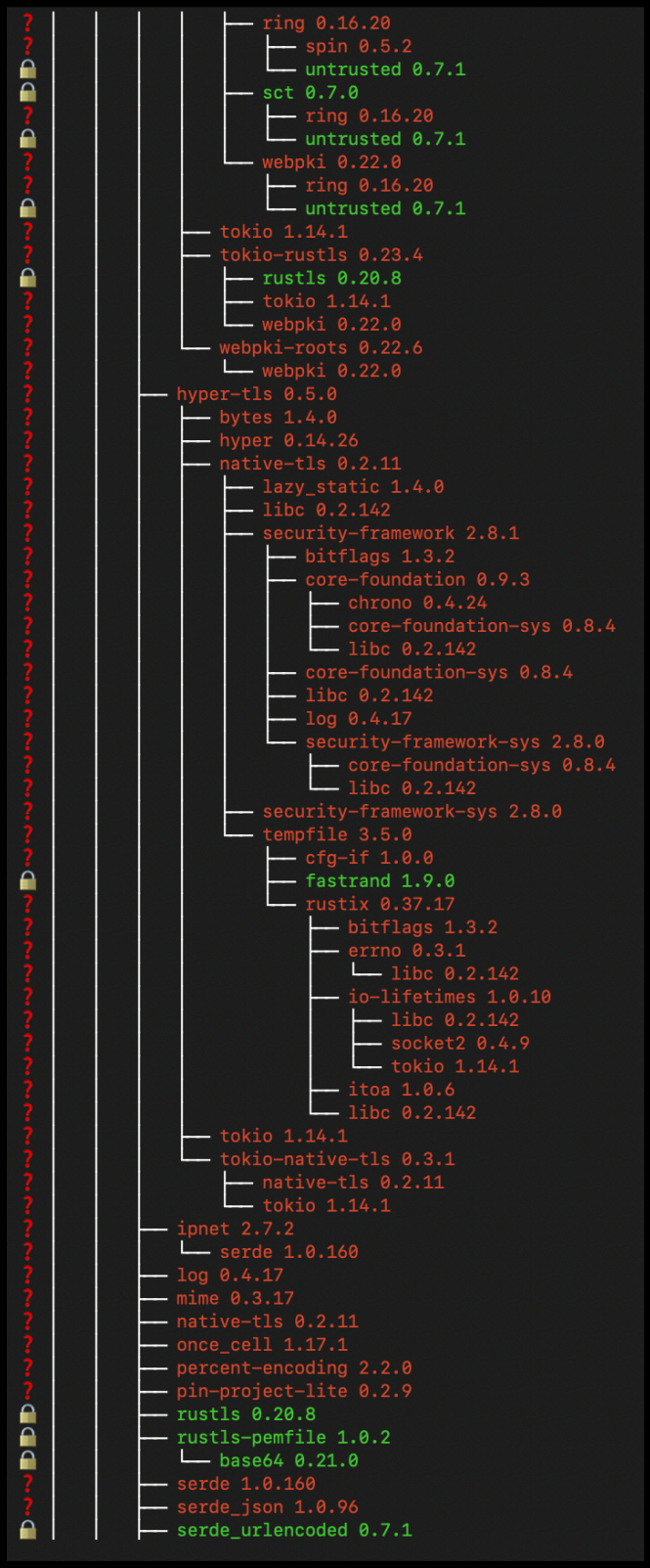
AUTOMATED TESTING



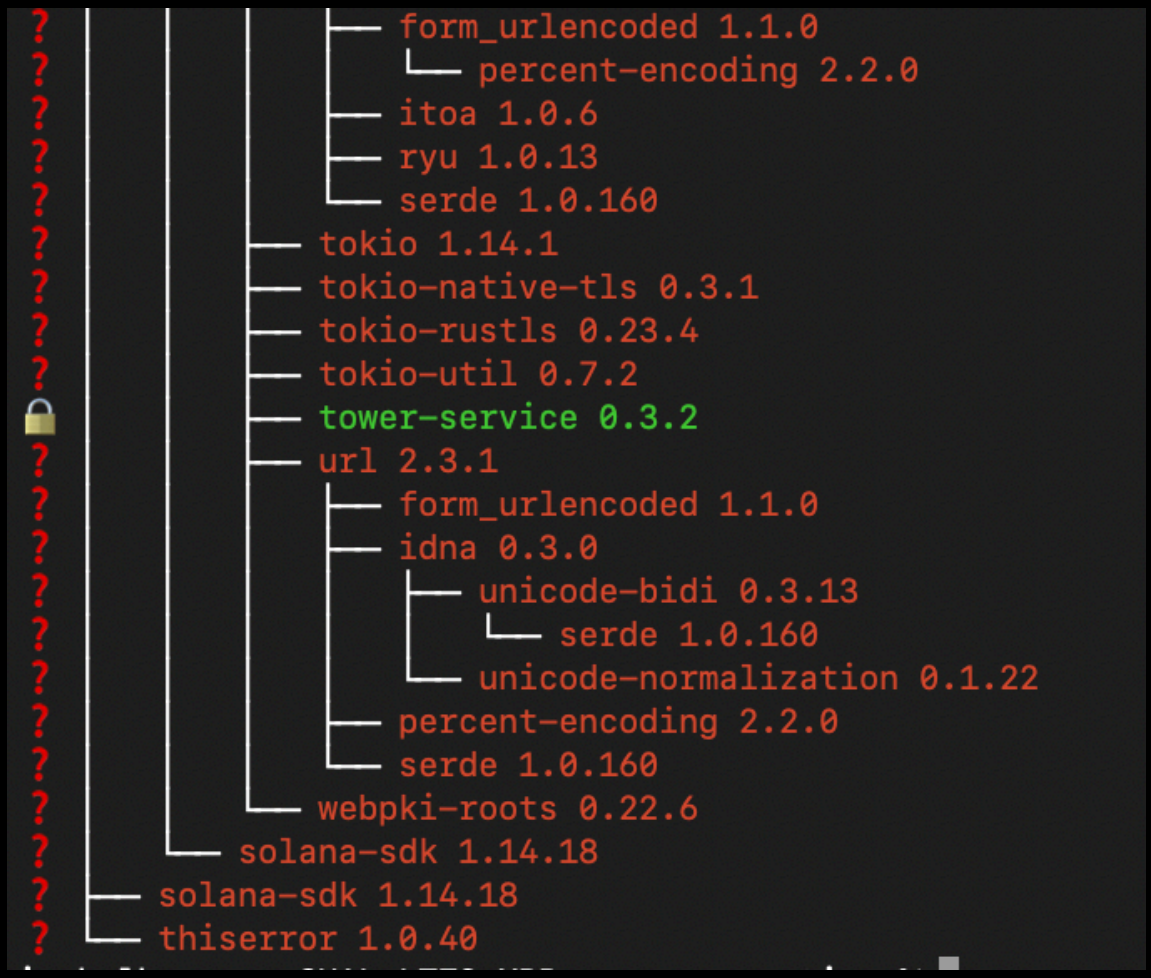
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN