Neon Labs - EVM v0.14.1

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 10th, 2022 - November 28th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	9/13/2022	Isabel Burruezo
0.2	Final Draft	11/27/2022	Isabel Burruezo
0.3	Draft Review	11/28/2022	Piotr Cielas
0.4	Draft Review	11/28/2022	Gabi Urrutia
1.0	Remediation Plan	12/06/2022	Isabel Burruezo
1.1	Remediation Plan Review	12/07/2022	Piotr Cielas
1.2	Remediation Plan Review	12/07/2022	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Neon Labs engaged Halborn to conduct a security audit on their program, beginning on October 10th, 2022 and ending on November 28th, 2022 .

Neon EVM is a tool that allows for Ethereum-like transactions to be processed on Solana, taking full advantage of the functionality native to Solana, including parallel execution of transactions. As such, Neon EVM allows dApps to operate with the low gas fees, high transaction speed, and high throughput of Solana, as well as offering access to the growing Solana market.

The security audit was scoped to the programs provided in the neon-evm 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 seven weeks for the engagement and assigned a full-time security engineer to audit the security of the program in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and Solana program hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Identify potential security issues within the program

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly accepted and acknowledged by the Neon Labs team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the Solana program audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the audit:

- Research into the architecture, purpose, and use of the platform.
- Program manual code review and walkthrough to identify 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)
- Scanning for common Solana vulnerabilities (soteria)

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. Neon EVM loader

- Repository: neon-evm
- Commit ID: 3f9edb2478c96b3a33eb8a5fd500f368484cbda5
- Programs in scope:
    1. evm-loader (evm_loader/program)

Out-of-scope: External libraries, dependencies and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	4

LIKELIHOOD

IMPACT

		(HAL-01)		
(HAL-02) (HAL-03)				
	(HAL-04)			
(HAL-05) (HAL-06) (HAL-07) (HAL-08)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
EXTRA FEES CHARGED WHEN TRANSACTIONS ARE UNSUCCESSFUL	Medium	SOLVED - 10/13/2022
VERIFICATION OF THE MAIN TREASURY'S AUTHORITY MISSING	Low	RISK ACCEPTED
HARDCODED GOVERNANCE ADDRESSES	Low	RISK ACCEPTED
SUSCEPTIBLE TO ACCESS OUT-OF-BOUNDS	Low	RISK ACCEPTED
REDUNDANT ACCESS TO HOLDER ACCOUNT MANAGEMENT FEATURES	Informational	ACKNOWLEDGED
DUPLICATE INSTRUCTION HANDLERS	Informational	ACKNOWLEDGED
MISSING CARGO OVERFLOW CHECKS	Informational	ACKNOWLEDGED
POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) EXTRA FEES CHARGED WHEN TRANSACTIONS ARE UNSUCCESSFUL - MEDIUM

Description:

Holder accounts are used to store EVM transactions that cannot be sent in a Solana instruction because they do not fit in a UDP datagram. Instead, such transactions are first uploaded in chunks and saved to Holder accounts, and then those accounts are provided as the EVM transaction data source to Neon EVM.

If an EVM transaction is created with the gas limit value too low and the actual gas consumed by executing transaction_step_from_instruction is greater than gas_limit, the finalize function is called again with the remaining gas value as a parameter (gas_used). However, gas_used is updated on line 158, with the value obtained from **gasometer.used_gas();**. This way, the calculated total_used_gas is the same, so the finalize function is called continuously exceeding the depth of the BPF call, instead of returning an error. This could result in additional fees being charged to the sender of the transaction for the redundant calculation.

Code Location:

Listing 1: src/instruction/transaction_step_from_account.rs (Line 75)

```
71 if let Some(gas_multiplier) = gas_multiplier {
72         storage.gas_limit = storage.gas_limit.saturating_mul(
↳ gas_multiplier);
73     }
74
75 do_begin(step_count, accounts, storage, account_storage, trx,
↳ caller, alt_cost)
```

Listing 2: src/instruction/transaction.rs (Line 69)

```
63             (Some((result, exit_reason, None)), executor.
↳ take_gasometer())
64         }
65         Err(e) => return Err(e)
66     }
67 };
68
69 finalize(accounts, storage, account_storage, results, gasometer.
↳ used_gas(), gasometer)
70 }
```

Listing 3: src/instruction/transaction.rs (Lines 169,173)

```
143 fn finalize<'a>(
144     accounts: Accounts<'a>,
145     mut storage: State<'a>,
146     account_storage: &mut ProgramAccountStorage<'a>,
147     results: Option<EvmResults>,
148     mut used_gas: U256,
149     mut gasometer: Gasometer,
150 ) -> ProgramResult {
151     debug_print!("finalize");
152
153     let accounts_operations = match results {
154         None => vec![],
155         Some((_, _, ref actions)) => {
156             let accounts_operations = account_storage.
↳ calc_accounts_operations(actions);
157             gasometer.record_accounts_operations(&
↳ accounts_operations);
158             used_gas = gasometer.used_gas();
159
160             accounts_operations
161         },
162     };
163
164     // The only place where checked math is required.
165     // Saturating math should be used everywhere else for gas
↳ calculation.
166     let total_used_gas = storage.gas_used.checked_add(used_gas);
167
168     // Integer overflow or more than gas_limit. Consume remaining
```

```
 ↳ gas and revert transaction with Out of Gas
169     if total_used_gas.is_none() || (total_used_gas > Some(storage.
 ↳ gas_limit)) {
170         let out_of_gas = Some((vec![], ExitError::OutOfGas.into(),
 ↳  None));
171         let remaining_gas = storage.gas_limit.saturating_sub(
 ↳ storage.gas_used);
172 
173         return finalize(accounts, storage, account_storage,
 ↳ out_of_gas, remaining_gas, gasometer);
174     }
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to return an error if total_used_gas is greater than the gas limit value, and to remove the finalize function call.

Remediation Plan:

SOLVED: The Neon Labs team solved this issue in commit 5bfb042c0908dd8f83d6f9f63b2c41aa1b1404f3: The code has been modified to remove recursion. Now, in case the calculated total_used_gas is greater than gas_limit, an error is returned. Also modified the gas calculation to update the value of total_used_gas.

3.2 (HAL-02) VERIFICATION OF THE MAIN TREASURY AUTHORITY MISSING - LOW

Description:

Operator accounts forward wrapped Ethereum transactions to Neon EVM. They charge transaction senders a NEON fee, and on each transaction execution they pay a fixed SOL fee to the Neon treasury.

CollectTreasure instruction can be called by anyone, and it allows transferring all the funds, except those necessary to be rent exempt, from Neon treasury to the main treasury. The MainTreasury account is an SPL Token account; however, its authority is not checked, leaving a small window of time open in which a random user could create a new account with the hardcoded seeds and transfer Neon treasury funds to it.

Code Location:

Listing 4: src/account/treasury.rs

```
43 impl<'a> MainTreasury<'a> {
44     pub fn from_account(info: &'a AccountInfo<'a>) -> Result<Self,
↳  ProgramError> {
45
46         let expected_key = Pubkey::create_with_seed(
47             &collateral_pool_base::id(),
48             collateral_pool_base::MAIN_BALANCE_SEED,
49             &spl_token::id())?;
50
51         if *info.key != expected_key {
52             return Err!(ProgramError::InvalidArgument; "Account {}
↳  - invalid main treasure account", info.key);
53         }
54
55         if *info.owner != spl_token::id() {
56             return Err!(ProgramError::InvalidArgument; "Account {}
↳  - invalid owner", info.key);
```

```
57          }
58
59          let account = spl_token::state::Account::unpack(&info.data
↳ .borrow())?;
60          if account.mint != spl_token::native_mint::id() {
61              return Err!(ProgramError::InvalidArgument; "Account {}
↳  - not wrapped SOL spl_token account", info.key);
62          }
63
64          Ok(Self { info })
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to add a check to verify the authority of MainTreasury.

Remediation Plan:

Remediation Plan:

RISK ACCEPTED: The Neon Labs team accepted the risk of this finding. Since the newly added instruction, CreateMainTreasury, creates and checks that the current program upgrade authority calls it. Therefore, anyone can perform this transfer on behalf of the treasury owner by calling CollectTreasury to and from already created and rigidly defined accounts.

3.3 (HAL-03) HARDCODED GOVERNANCE ADDRESSES - LOW

Description:

Important governance account addresses are hardcoded in evm_loader/program/src/config.rs. In case those addresses are compromised, the program authority has to redeploy the program to update them.

Code Location:

Listing 5: evm_loader/program/src/config.rs

```
21 // NOTE: when expanding this list, add same addresses to the
22 // alpha configuration as well
23 operators_whitelist![
24     "NeonPQFrw5stVvs1rFLDxALWUBDCnSPsWBP83RfNUKK",
25     "NeoQM3utcHGxhKT41Nq81g8t4xGcPNFpkAgYj1N2N8v",
26     "Gw3Xiwve6HdvpJeQguhwT23cpK9nRjSy1NpNYCFY4XU9",
27     "DSRVyWpSVLEcHih9CVND2aGNBZxNW5bt34GEaK4aDk5i",
28 ];
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider making those governance addresses mutable and implement a function to update them in case they are compromised.

Remediation Plan:

RISK ACCEPTED: The Neon Labs team accepted the risk of this finding.

3.4 (HAL-04) SUSCEPTIBLE TO ACCESS OUT-OF-BOUNDS - LOW

Description:

The EvmInstruction::CreateAccountV03 instruction handler expects the transaction sender to provide 3 required accounts and one extra:
- operator account
- system program account
- ethereum account

Those accounts are retrieved from the AccountInfo slice by directly accessing the slice at relevant indices. Because the length of AccountInfo is not validated, the program may end up accessing the slice out-of-bounds if the transaction sender provides fewer accounts than expected and trigger a crash.

The instruction data processed by this handler is also susceptible to out-of-bounds access because the program assumes the provided slice is of required length when it generates the array reference.

This issue was also identified in the HolderCreate, HolderDelete, HolderWrite and CollectTreasure instruction handler in parsing input accounts and instruction data.

Code Location:

Listing 6: evm_loader/program/src/instruction/transaction_step_from_account_no_chainid.rs (Lines 19,21)

```
10 struct Accounts<'a> {
11     operator: Operator<'a>,
12     system_program: program::System<'a>,
13     ether_account: &'a AccountInfo<'a>,
14 }
15
16 pub fn process<'a>(program_id: &'a Pubkey, accounts: &'a [
↳ AccountInfo<'a>], instruction: &[u8]) -> ProgramResult {
```

```
17      solana_program::msg!("Instruction: Create Account");
18      let parsed_accounts = Accounts {
19          operator: unsafe { Operator::from_account_not_whitelisted
↳ (&accounts[0]) }?,
20          system_program: program::System::from_account(&accounts
↳ [1])?,
21          ether_account: &accounts[2],
22      };
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider retrieving with the solana_program::account_info::next_account_info utility function to prevent Rust panics when accessing OOB.

Remediation Plan:

RISK ACCEPTED: The Neon Labs team accepted the risk of this finding.

3.5 (HAL-05) REDUNDANT ACCESS TO HOLDER ACCOUNT MANAGEMENT FEATURES - INFORMATIONAL

Description:

Holder accounts are used to store such EVM transactions that cannot be sent in one Solana instruction because they do not fit in one UDP datagram. Instead, such transactions are first uploaded in chunks and saved in Holder accounts and then those accounts are provided as EVM transaction data source to Neon EVM.

The HolderCreate instruction allows any unauthorized operator to create the Holder account with the operator as owner and setting the trasanction_hash.

The HolderWrite can be called by the owner, and it is used to upload the transaction in chunks and save them in it, as mentioned before.

The HolderDelete instruction allows the owner to delete the Holder account.

Although any operator can create and write to and delete their Holder account, only authorized operators can execute Neon EVM transactions from their Holder accounts. Therefore, at this project development stage only authorized operators should be allowed to manage Holder accounts in order not to confuse the users of the program.

Code Location:

Listing 7: evm_loader/program/src/instruction/account_holder_create.rs (Lines 12,15)

```
11 let holder = &accounts[0];
12 let operator = unsafe { Operator::from_account_not_whitelisted(&
↳ accounts[1]) }?;
13
14 Holder::init(holder, crate::account::holder::Data {
15     owner: *operator.key,
16     transaction_hash: [0_u8; 32]
17 })?;
```

Listing 8: evm_loader/program/src/instruction/account_holder_write.rs (Line 33)

```
33 let operator = unsafe { Operator::from_account_not_whitelisted(&
↳ accounts[1]) }?;
34
35 holder.validate_owner(&operator)?;
```

Listing 9: evm_loader/program/src/instruction/account_holder_delete.rs (Line 12)

```
11 let holder = Holder::from_account(program_id, &accounts[0])?;
12 let operator = unsafe { Operator::from_account_not_whitelisted(&
↳ accounts[1]) }?;
13
14 holder.validate_owner(&operator)?;
```

Listing 10: evm_loader/program/src/account/operator.rs (Line 21)

```
11 pub fn from_account(info: &'a AccountInfo<'a>) -> Result<Self,
↳ ProgramError> {
12         let is_authorized = crate::config::
↳ AUTHORIZED_OPERATOR_LIST
13             .binary_search(info.key).is_ok();
14
15         if !is_authorized {
16             return Err!(EvmLoaderError::UnauthorizedOperator.into
```

```
↳ (); "Account {} - expected authorized operator", info.key);
17          }
18
19          unsafe { Self::from_account_not_whitelisted(info) }
20  }
21  pub unsafe fn from_account_not_whitelisted(info: &'a AccountInfo<'
↳ a>) -> Result<Self, ProgramError> {
22          if !solana_program::system_program::check_id(info.owner) {
23              return Err!(ProgramError::InvalidArgument; "Account {}
↳  - expected system owned", info.key);
24          }
25
26          if !info.is_signer {
27              return Err!(ProgramError::InvalidArgument; "Account {}
↳  - expected signer", info.key);
28          }
29
30          if info.data_len() > 0 {
31              return Err!(ProgramError::InvalidArgument; "Account {}
↳  - expected empty", info.key);
32          }
33
34          Ok(Self { info })
```

Listing 11: evm_loader/program/src/instruction/transaction_step_from_account.rs (Lines 25,35)

```
24  let accounts = Accounts {
25          operator: Operator::from_account(&accounts[1])?,
26          treasury: Treasury::from_account(program_id,
↳ treasury_index, &accounts[2])?,
27          operator_ether_account: EthereumAccount::from_account(
↳ program_id, &accounts[3])?,
28          system_program: program::System::from_account(&accounts
↳ [4])?,
29          neon_program: program::Neon::from_account(program_id, &
↳ accounts[5])?,
30          remaining_accounts: &accounts[6..]
31      };
32
33      let mut account_storage = ProgramAccountStorage::new(
34          program_id,
35          &accounts.operator,
```

```
36          Some(&accounts.system_program),
37          accounts.remaining_accounts,
38      )?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to restrict access to the HolderCreate, HolderWrite and HolderDelete instructions to authorized operators only.

Remediation Plan:

ACKNOWLEDGED: The Neon Labs team acknowledged this finding.

3.6 (HAL-06) DUPLICATE INSTRUCTION HANDLERS - INFORMATIONAL

Description:

The evm-loader can process Ethereum transactions pre-saved in dedicated Holder accounts. The TransactionStepFromAccountNoChainId and TransactionStepFromAccount instruction handlers verify if the provided transaction is a fresh one or is it already being processed.

The only difference between those two handlers is the TransactionStepFromAccountNoChai handler sets a fixed transaction gas limit, crate::config::GAS_LIMIT_MULTIPLIER_NO_CHAINID. The code can be optimized to decrease the program size and lower deployment and maintenance cost.

Code Location:

Listing 12: evm_loader/program/src/instruction/transaction_step_from_account_no_chainid.rs (Lines 38,39,40)

```
13 pub fn process<'a>(program_id: &'a Pubkey, accounts: &'a [
↳ AccountInfo<'a>], instruction: &[u8]) -> ProgramResult {
14     solana_program::msg!("Instruction: Begin or Continue
↳ Transaction from Account Without ChainId");
15
16     let treasury_index = u32::from_le_bytes(*array_ref![
↳ instruction, 0, 4]);
17     let step_count = u64::from(u32::from_le_bytes(*array_ref![
↳ instruction, 4, 4]));
18     let alt_gas_used = alt_cost(accounts.len() as u64);
19
20     let holder_or_storage_info = &accounts[0];
21
22     let accounts = Accounts {
23         operator: Operator::from_account(&accounts[1])?,
24         treasury: Treasury::from_account(program_id,
↳ treasury_index, &accounts[2])?,
25         operator_ether_account: EthereumAccount::from_account(
↳ program_id, &accounts[3])?,
```

```
26          system_program: program::System::from_account(&accounts
↳ [4])?,
27          neon_program: program::Neon::from_account(program_id, &
↳ accounts[5])?,
28          remaining_accounts: &accounts[6..]
29      };
30  let mut account_storage = ProgramAccountStorage::new(
31          program_id,
32          &accounts.operator,
33          Some(&accounts.system_program),
34          accounts.remaining_accounts,
35      )?;
36
37      let gas_multiplier = U256::from(
↳ GAS_LIMIT_MULTIPLIER_NO_CHAINID);
38      super::transaction_step_from_account::execute(
39          program_id, holder_or_storage_info, accounts, &mut
↳ account_storage, step_count, Some(gas_multiplier), alt_gas_used
40      )
41
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider removing the TransactionStepFromAccountNoChainId instruction and its handler and modify the TransactionStepFromAccount instruction handler to a case when the transaction is missing chain ID:

Listing 13

```
1 let mut storage = State::new(program_id, holder_or_storage_info, &
↳ accounts, caller, &trx)?;
2
3 if trx.chain_id.is_none() {
4     storage.gas_limit = storage.gas_limit.saturating_mul(U256::
↳ from(crate::config::GAS_LIMIT_MULTIPLIER_NO_CHAINID));
```

```
5 }
```

Remediation Plan:

ACKNOWLEDGED: The Neon Labs team acknowledged this finding.

3.7 (HAL-07) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL

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

Code Location:

- evm_loader/program/Cargo.toml

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation Plan:

Remediation Plan:

ACKNOWLEDGED: The Neon Labs team acknowledged this finding.

3.8 (HAL-08) 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:

Listing 14

```
src/precompile/query_account.rs:42:    let method_id: &[u8; 4] =
↳ method_id.try_into().unwrap_or(&[0_u8; 4]);
src/precompile/spl_token.rs:150:    U256::from_big_endian_fast(
↳ arrayref::array_ref![input, 0, 32]).try_into().unwrap()
src/precompile/spl_token.rs:189:        let allocate =
↳ system_instruction::allocate(&account.key, space.try_into().unwrap
↳ ());
src/precompile/spl_token.rs:201:            space.try_into().
↳ unwrap(),
src/precompile/spl_token.rs:248:        &mint_authority.unwrap_or(
↳ signer_pubkey),
src/precompile/spl_token.rs:249:        Some(&freeze_authority.
↳ unwrap_or(signer_pubkey)),
src/precompile/spl_token.rs:295:        &owner.unwrap_or(
↳ signer_pubkey)
src/precompile/spl_token.rs:603:    *delegate = token.delegate.map
↳ (Pubkey::to_bytes).unwrap_or_default();
src/precompile/spl_token.rs:605:    *close_authority = token.
↳ close_authority.map(Pubkey::to_bytes).unwrap_or_default();
src/precompile/spl_token.rs:633:    *freeze_authority = mint.
↳ freeze_authority.map(Pubkey::to_bytes).unwrap_or_default();
```

```
11 src/precompile/spl_token.rs:634:    *mint_authority = mint.
↳ mint_authority.map(Pubkey::to_bytes).unwrap_or_default();
12 src/precompile/neon_token.rs:43:    let method_id: &[u8; 4] =
↳ method_id.try_into().unwrap_or(&[0_u8; 4]);
13 src/precompile/neon_token.rs:130:    ).unwrap();
14 src/external_programs/spl_associated_token.rs:33:        let mut
↳ funder = accounts.get_mut(funder_key).unwrap();
15 src/external_programs/spl_associated_token.rs:42:        let mut
↳ associated_token_account = accounts.get_mut(
↳ associated_token_account_key).unwrap();
16 src/external_programs/system.rs:15:    let system_instruction:
↳ SystemInstruction = bincode::deserialize(instruction).unwrap();
17 src/external_programs/system.rs:22:                let mut funder
↳ = accounts.get_mut(funder_key).unwrap();
18 src/external_programs/system.rs:31:                let mut account
↳  = accounts.get_mut(account_key).unwrap();
19 src/external_programs/system.rs:43:            let mut account =
↳ accounts.get_mut(account_key).unwrap();
20 src/external_programs/system.rs:56:                let mut from =
↳ accounts.get_mut(from_key).unwrap();
21 src/external_programs/system.rs:69:                let mut to =
↳ accounts.get_mut(to_key).unwrap();
22 src/external_programs/system.rs:75:            let account =
↳ accounts.get_mut(account_key).unwrap();
23 src/external_programs/metaplex.rs:49:        let metadata_account
↳ = accounts.get_mut(metadata_account_key).unwrap();
24 src/external_programs/metaplex.rs:59:        let mint_info =
↳ accounts.get_mut(mint_key).unwrap().into_account_info();
25 src/external_programs/metaplex.rs:103:        let metadata_account
↳  = accounts.get_mut(metadata_account_key).unwrap();
26 src/external_programs/metaplex.rs:123:        let metadata_info =
↳ accounts.get_mut(metadata_account_key).unwrap().into_account_info
↳ ();
27 src/external_programs/metaplex.rs:128:        let mint_info =
↳ accounts.get_mut(mint_key).unwrap().into_account_info();
28 src/external_programs/metaplex.rs:141:        let
↳ update_authority_info = accounts.get_mut(update_authority_key).
↳ unwrap().into_account_info();
29 src/external_programs/metaplex.rs:153:        let edition_account
↳ = accounts.get_mut(edition_account_key).unwrap();
30 src/external_programs/metaplex.rs:163:        let metadata_account
↳  = accounts.get_mut(metadata_account_key).unwrap();
31 src/external_programs/metaplex.rs:175:        let mint_account =
↳ accounts.get_mut(mint_key).unwrap();
```

```
32 src/instruction/transaction.rs:184:        let apply_state =
↳ apply_state.unwrap_or_else(
33 src/instruction/transaction_execute_from_instruction.rs:137:
↳ let apply_state = apply_state.unwrap_or_else(
34 src/executor/state.rs:92:        let actions_len = self.stack.pop
↳ ().unwrap_or(0);
35 src/executor/state.rs:249:         known_storage.unwrap_or_else(||
↳ self.backend.storage(from_address, from_key))
36 src/executor/gasometer.rs:39:             paid_gas: paid_gas.
↳ unwrap_or(U256::zero()),
37 src/executor/machine.rs:63:         self.runtime.serialize(&mut &
↳ mut buffer).unwrap();
38 src/executor/machine.rs:64:         self.executor.state.serialize(&
↳ mut &mut buffer).unwrap();
39 src/executor/machine.rs:71:         let runtime = BorshDeserialize
↳ ::deserialize(&mut buffer).unwrap();
40 src/executor/machine.rs:72:         let state = ExecutorState::
↳ deserialize(&mut buffer, backend).unwrap();
41 src/executor/machine.rs:268:         let (runtime, _) = self.
↳ runtime.last_mut().unwrap();
42 src/account_storage/backend.rs:40:            let offset: usize =
↳ (8 + (clock_slot - 1 - number.as_u64()) * 40).try_into().unwrap();
43 src/account_storage/backend.rs:109:            .unwrap_or_else(||
↳ panic!("Account {} - storage account not found", solana_address));
44 src/account_storage/backend.rs:112:            let storage =
↳ EthereumStorage::from_account(self.program_id, account).unwrap();
45 src/account_storage/base.rs:107:        self.ethereum_accounts.
↳ get_mut(address).unwrap() // mutable accounts always present
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to use the unwrap function in the production environment because its use causes panic! and may crash the contract without verbose error messages. Crashing the system will result in a loss of availability and, in some cases, even private information stored

in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrapping, or using the error-chain crate for errors.

Remediation Plan:

ACKNOWLEDGED: The Neon Labs team acknowledged this finding.

AUTOMATED TESTING

4.1 AUTOMATED VULNERABILITY SCANNING

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was Soteria, a security analysis service for Solana programs. Soteria performed a scan on all the programs and sent the compiled results to the analyzers to locate any vulnerabilities.

Results:

The issues identified were verified to be false positives.

```
anchor_lang_version: =3.5.0 anchorVersionTooOld: 0
 - ✓ [00m:02s] Loading IR From File
 - * [00m:00s] Running Compiler Optimization Passes
EntryPoints:
entrypoint
 - ✓ [00m:00s] Running Compiler Optimization Passes
 - ✓ [00m:01s] Running Pointer Analysis
============This account may be UNTRUSTFUL!===============
Found a potential vulnerability at line 60, column 24 in program/src/account/holder.rs
The account info is not trustful:

 54|
 55|        let mut data = self.info.data.borrow_mut();
 56|        data[Self::SIZE..].fill(0);
 57|    }
 58|
 59|    pub fn write(&mut self, offset: usize, bytes: &[u8]) {
>60|        let mut data = self.info.data.borrow_mut(); //len = 88
 61|
 62|
 63|        let begin = Self::SIZE + offset;   //983547....
 64|        let end = begin + bytes.len();  //983547....
 65|        solana_program::msg!("self size: {}", begin);
 66|
>>>Stack Trace:
>>>evm_loader::instruction::account_holder_write::process::h37d6e8d5a6c5f3b6 [program/src/entrypoint.rs:52]
>>>  evm_loader::account::holder::_$LT$impl$u20$evm_loader..account..AccountData$LT$evm_loader..account..holder..Data$GT$$GT$::write::hf93deadbf7061c23 [program/src/instruction/account_holder_write.rs:54]
```

```
============This arithmetic operation may be UNSAFE!===============
Found a potential vulnerability at line 71, column 37 in program/src/account/ether_storage.rs
The sub operation may result in underflows:

 65|
 66|        let new_len = self.info.data_len() + 1 + 32; // new_len <= 8.25 kb
 67|        self.info.realloc(new_len, false)?;
 68|
 69|        let minimum_balance = Rent::get()?.minimum_balance(new_len);
 70|        if self.info.lamports() < minimum_balance {
>71|            let required_lamports = minimum_balance - self.info.lamports();
 72|            system.transfer(operator, self.info, required_lamports)?;
 73|        }
 74|
 75|        let mut data = self.info.data.borrow_mut();
 76|        let data = &mut data[1..]; // skip tag
 77|
>>>Stack Trace:
>>>evm_loader::instruction::transaction_execute_from_instruction::process::hb5a9aaa84176e693 [program/src/entrypoint.rs:61]
>>>  evm_loader::instruction::transaction_execute_from_instruction::execute::h1577b3be3933e6aa [program/src/instruction/transaction_execute_from_instruction.rs:54]
>>>    evm_loader::account_storage::apply::_$LT$impl$u20$evm_loader..account_storage..ProgramAccountStorage$GT$::apply_state_change::h31e4ebde0645ddf8 [program/src/instruction/transaction_execute_from_ins
truction.rs:141]
>>>      evm_loader::account_storage::apply::_$LT$impl$u20$evm_loader..account_storage..ProgramAccountStorage$GT$::update_storage_infinite::he85286@ad4563275 [program/src/account_storage/apply.rs:133]
>>>        evm_loader::account::ether_storage::_$LT$impl$u20$evm_loader..account..AccountData$LT$evm_loader..account..ether_storage..Data$GT$$GT$::set::h72901d6ead3da2b9 [program/src/account_storage/apply
.rs:298]

============This arithmetic operation may be UNSAFE!===============
Found a potential vulnerability at line 98, column 52 in program/src/state_account.rs
The sub operation may result in underflows:

 92|        }
 93|
 94|        let mut storage = State::from_account(program_id, info)?;
 95|        storage.check_accounts(accounts)?;
 96|
 97|        let clock = Clock::get()?;
>98|        if (*operator.key != storage.operator) && ((clock.slot - storage.slot) <= OPERATOR_PRIORITY_SLOTS) {
 99|            return Err!(ProgramError::InvalidAccountData; "operator.key != storage.operator");
100|        }
101|
102|        if storage.operator != *operator.key {
103|            storage.operator = *operator.key;
104|            storage.slot = clock.slot;
>>>Stack Trace:
>>>evm_loader::instruction::transaction_step_from_instruction::process::ha393b69cd6d111cb [program/src/entrypoint.rs:64]
>>>  evm_loader::state_account::_$LT$impl$u20$evm_loader..account..AccountData$LT$evm_loader..account..state..Data$GT$$GT$::restore::hf1319e391194d3f5 [program/src/instruction/transaction_step_from_instru
ction.rs:51]

 - ✓ [00m:05s] Building Static Happens-Before Graph
 - ✓ [00m:00s] Detecting Vulnerabilities
detected 4 untrustful accounts in total.
detected 13 unsafe math operations in total.
```

AUTOMATED TESTING

4.2 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo-audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. 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:

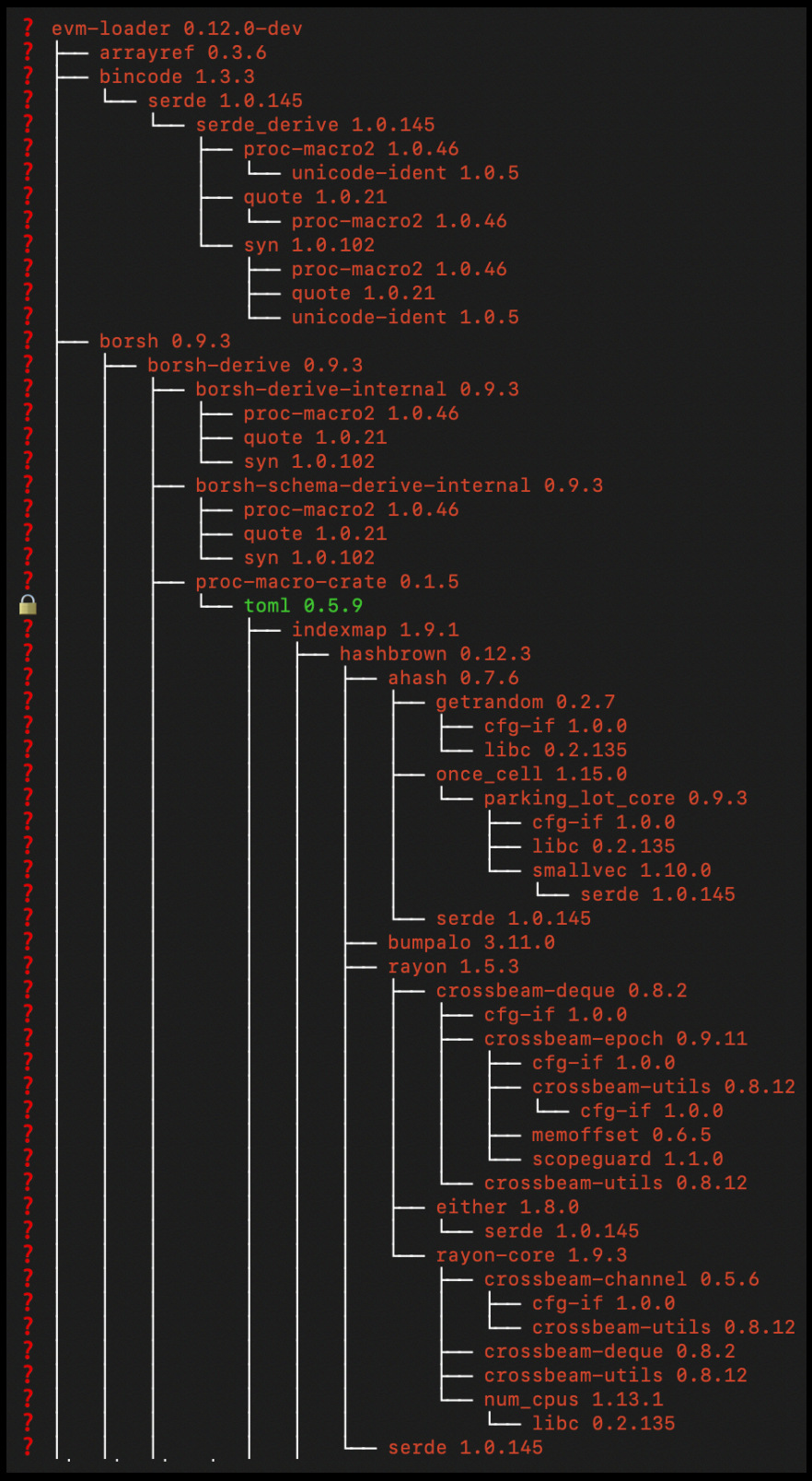
ID	package	Short Description
RUSTSEC-2020-0071	time	Potential segfault in the time crate

4.3 UNSAFE RUST CODE DETECTION

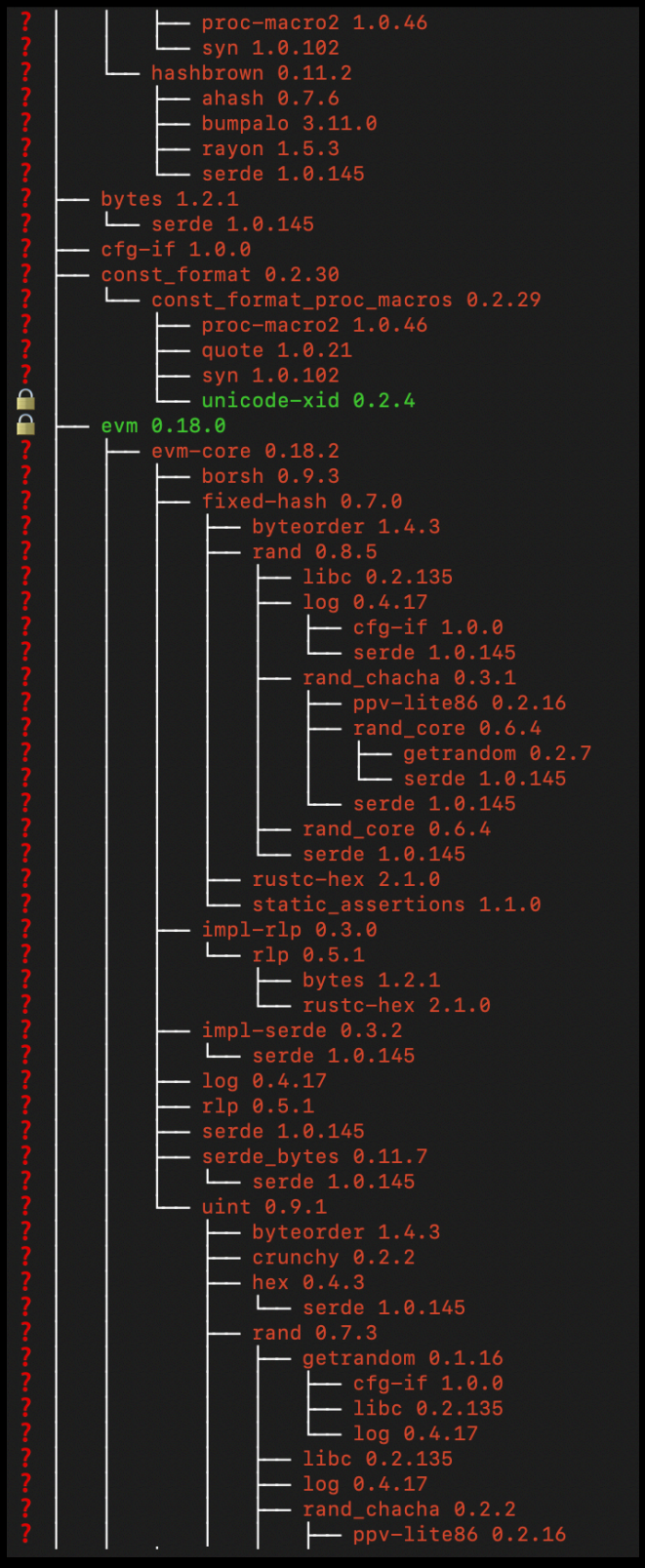
Description:

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:

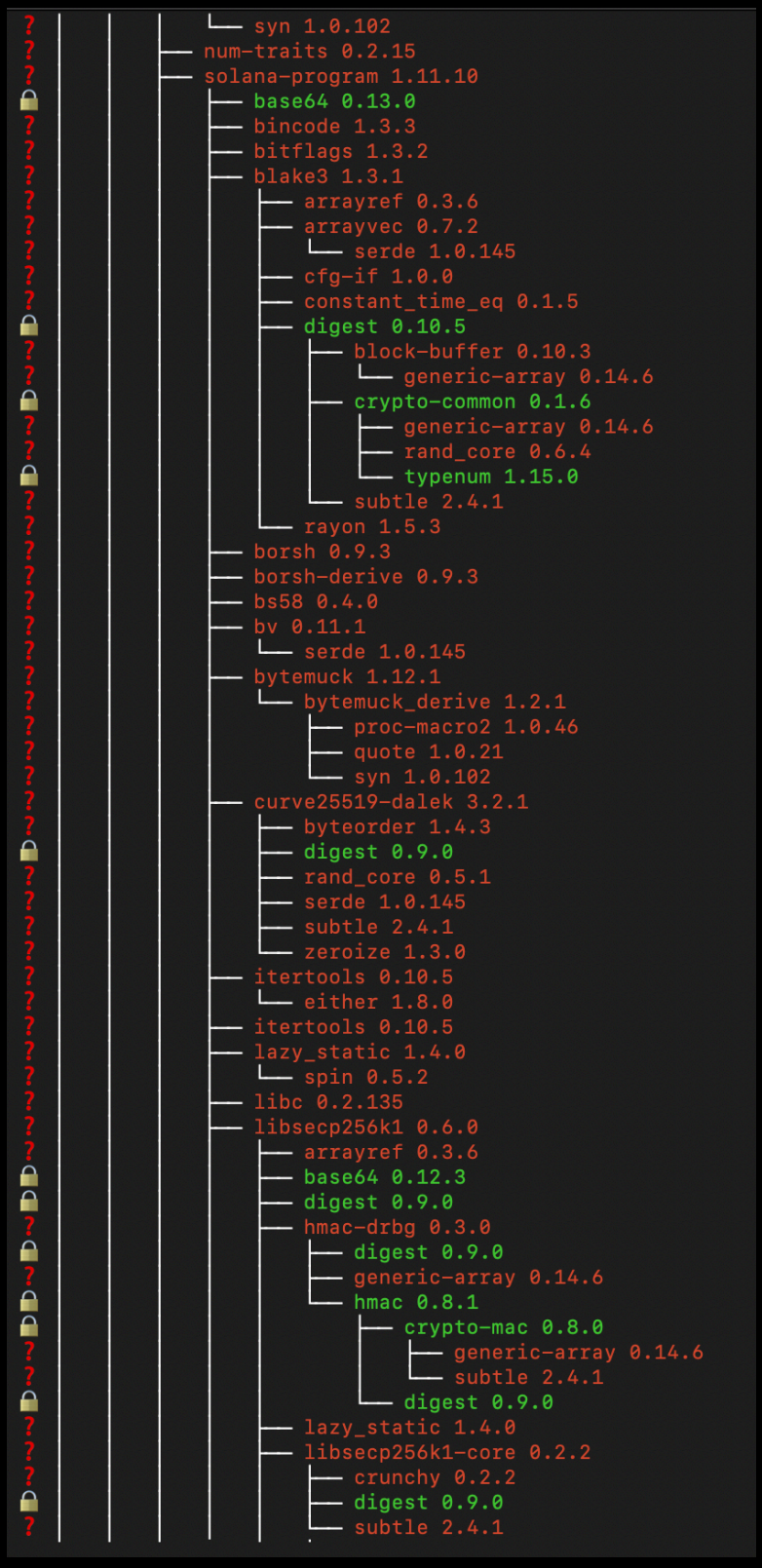

evm-loader 0.12.0-dev
arrayref 0.3.6
bincode 1.3.3
serde 1.0.145
serde_derive 1.0.145
proc-macro2 1.0.46
unicode-ident 1.0.5
quote 1.0.21
proc-macro2 1.0.46
syn 1.0.102
proc-macro2 1.0.46
quote 1.0.21
unicode-ident 1.0.5
borsh 0.9.3
borsh-derive 0.9.3
borsh-derive-internal 0.9.3
proc-macro2 1.0.46
quote 1.0.21
syn 1.0.102
borsh-schema-derive-internal 0.9.3
proc-macro2 1.0.46
quote 1.0.21
syn 1.0.102
proc-macro-crate 0.1.5
toml 0.5.9
indexmap 1.9.1
hashbrown 0.12.3
ahash 0.7.6
getrandom 0.2.7
cfg-if 1.0.0
libc 0.2.135
once_cell 1.15.0
parking_lot_core 0.9.3
cfg-if 1.0.0
libc 0.2.135
smallvec 1.10.0
serde 1.0.145
serde 1.0.145
bumpalo 3.11.0
rayon 1.5.3
crossbeam-deque 0.8.2
cfg-if 1.0.0
crossbeam-epoch 0.9.11
cfg-if 1.0.0
crossbeam-utils 0.8.12
cfg-if 1.0.0
memoffset 0.6.5
scopeguard 1.1.0
crossbeam-utils 0.8.12
either 1.8.0
serde 1.0.145
rayon-core 1.9.3
crossbeam-channel 0.5.6
cfg-if 1.0.0
crossbeam-utils 0.8.12
crossbeam-deque 0.8.2
crossbeam-utils 0.8.12
num_cpus 1.13.1
libc 0.2.135
serde 1.0.145

AUTOMATED TESTING

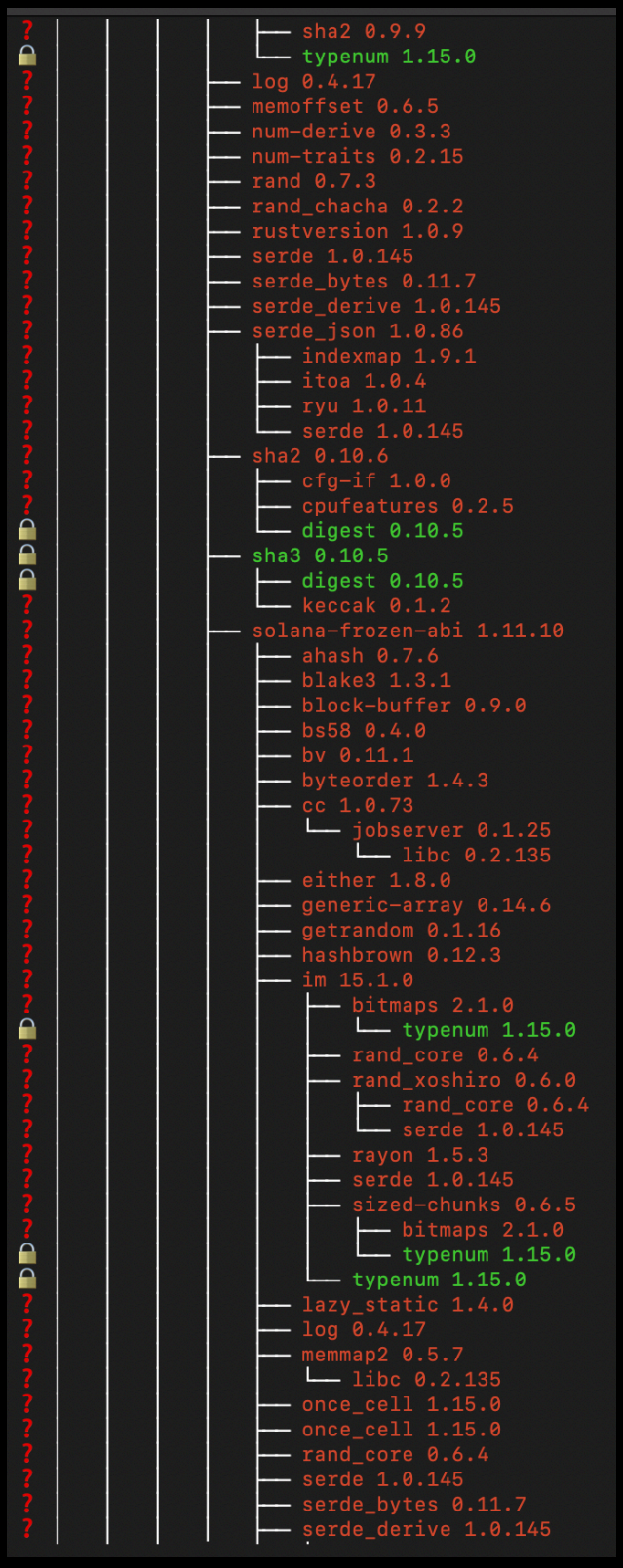


AUTOMATED TESTING

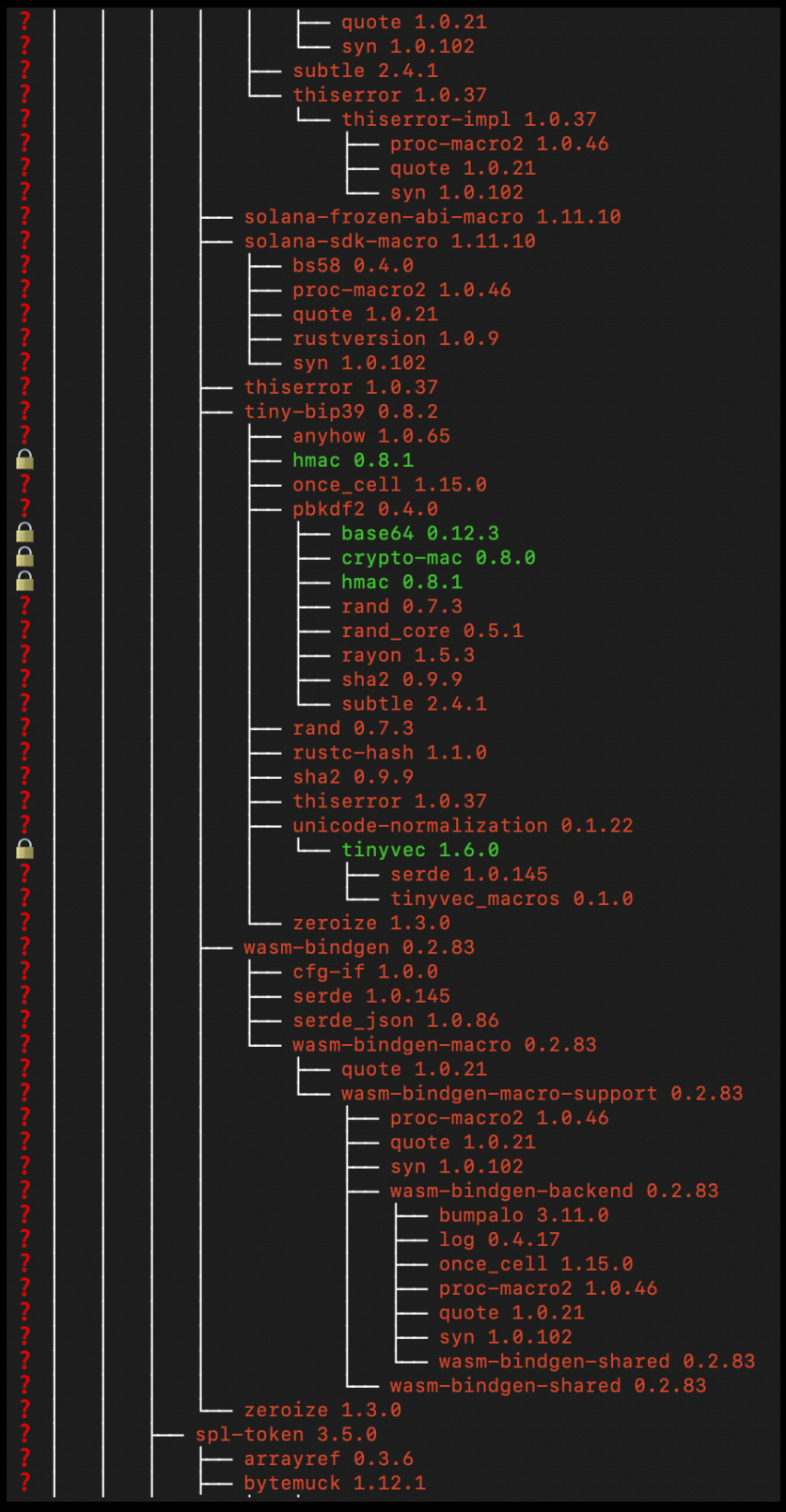
AUTOMATED TESTING



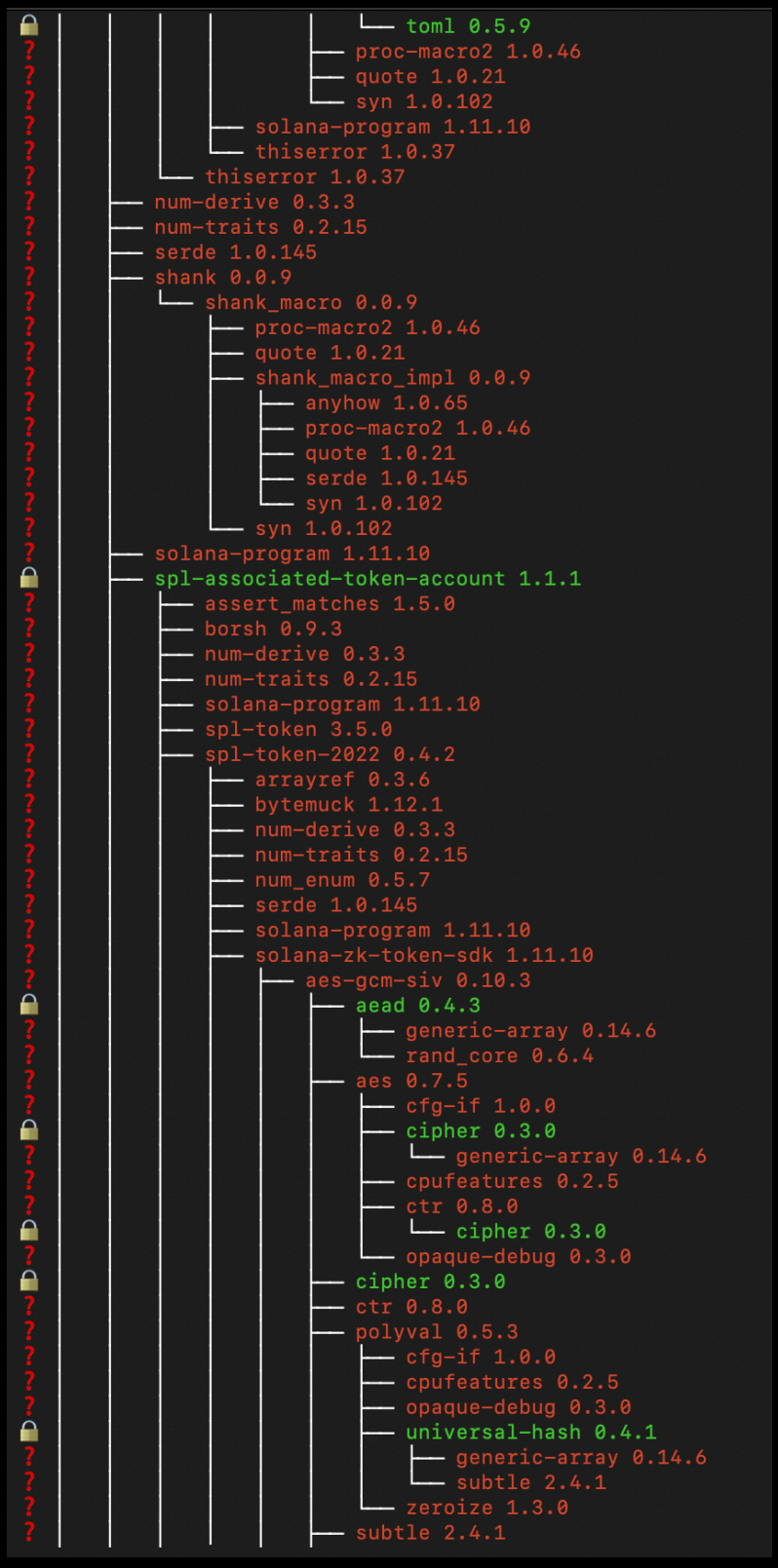
AUTOMATED TESTING



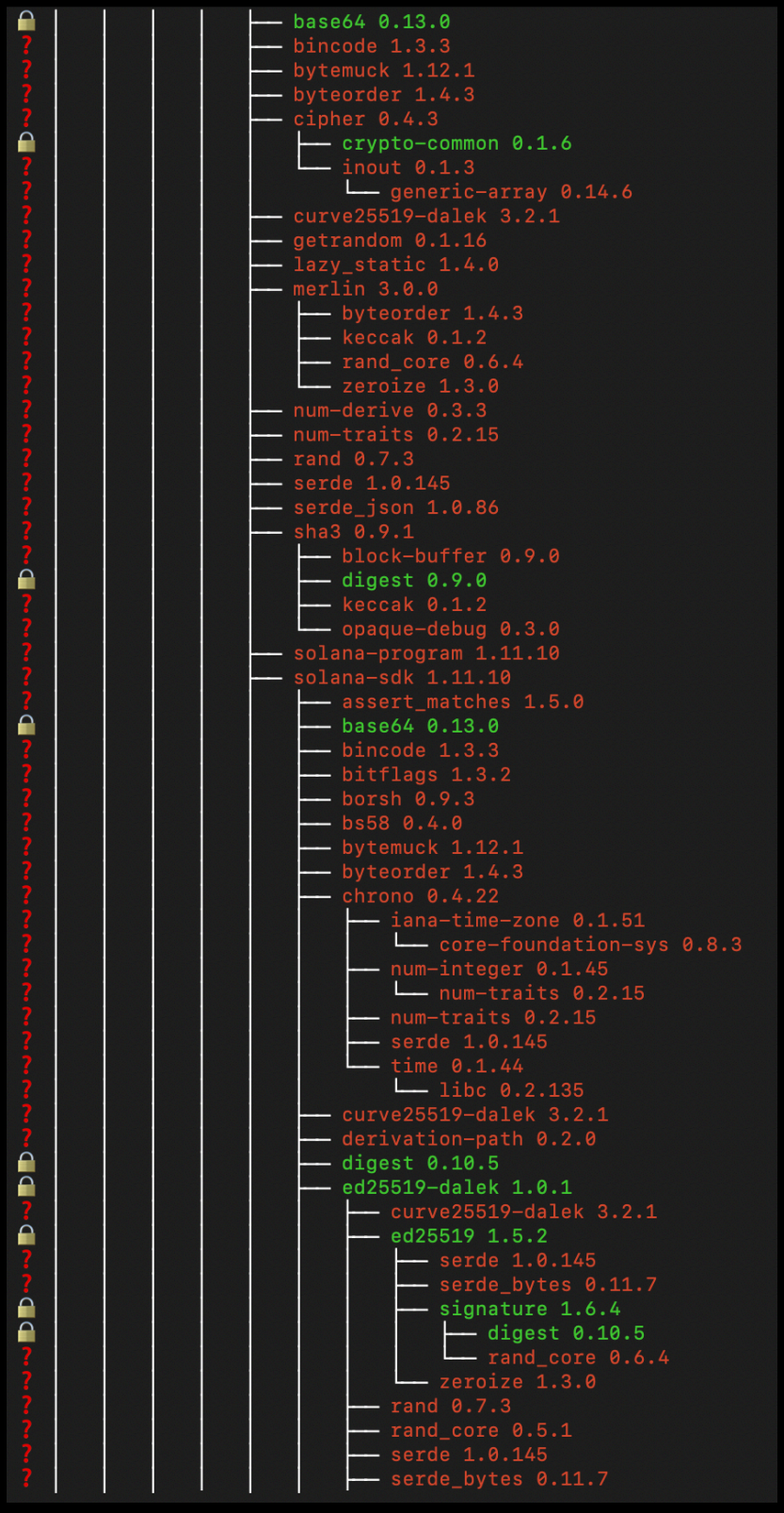
AUTOMATED TESTING



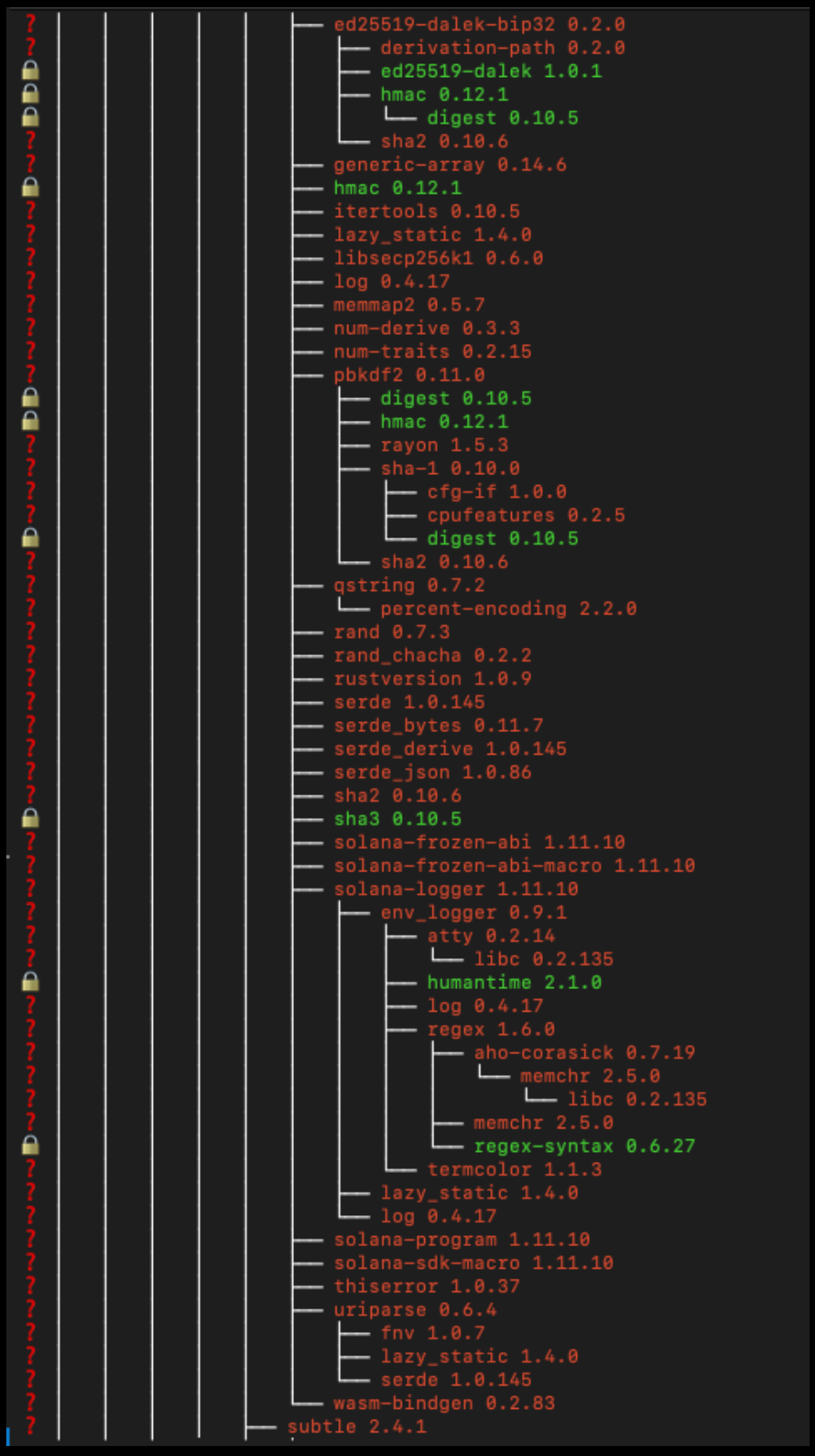
AUTOMATED TESTING



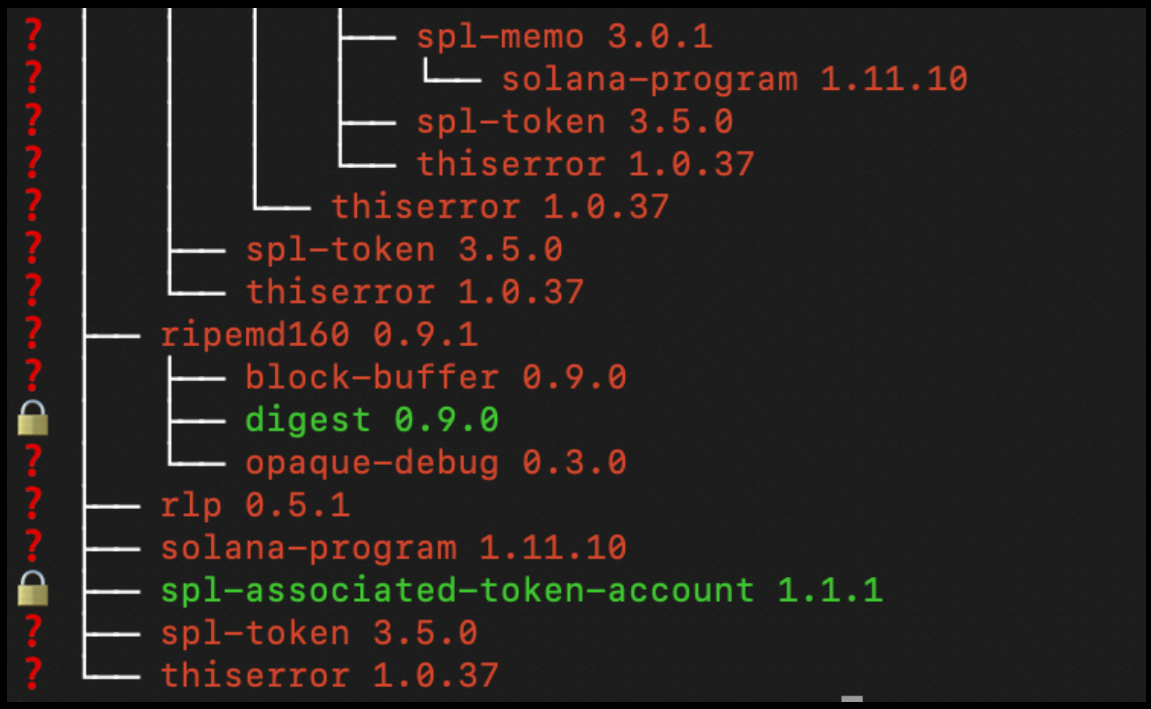
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN