DeBridge - Solana Events Reader

Rust Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 20th, 2023 - April 21st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/01/2023	Przemyslaw Swiatowiec
0.2	Document Updates	04/03/2023	Przemyslaw Swiatowiec
0.3	Final Draft	04/06/2023	Przemyslaw Swiatowiec
0.4	Draft Review	04/07/2023	Piotr Cielas
0.5	Draft Review	04/07/2023	Gabi Urrutia
1.0	Remediation Plan	04/17/2023	Przemyslaw Swiatowiec
1.1	Remediation Plan Review	04/21/2023	Isabel Burruezo
1.3	Remediation Plan Review	04/21/2023	Piotr Cielas
1.4	Remediation Plan Review	04/24/2023	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
Przemyslaw Swiatowiec	Halborn	Przemyslaw.Swiatowiec@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Solana Event Reader is an application that allows external clients (DeBridge Network Validator and DeBridge Backend services) to subscribe to gRPC streams and receive valid events captured from the DeBridge Solana contract. The application implements two services:
- Reader Service that connects to Solana node via WebSocket and RPC connection, captures events from DeBridge programs transactions and stores it in the SQL database,
- GRPC Service that serves events stored in the database using gRPC protocol.

DeBridge engaged Halborn to conduct a security audit on their Solana Event Reader application, beginning on March 20th, 2023 and ending on April 21st, 2023 . Halborn was provided access to the program's source code to conduct white-box security testing using tools to scan, detect, and validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

The security assessment was scoped to the programs provided in the debridge-solana-events-reader 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 5 weeks for the engagement and assigned a 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:

- Ensure that program functions operate as intended,
- Identify potential security issues with the programs.

In summary, Halborn identified some risks that were mostly addressed by the DeBridge team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the program audit. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help to 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).

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. DeBridge Solana Events Reader

- Repository: debridge-solana-events-reader
- Commit ID: 4e8b4df1da42048470e8b70ca450dc6de4f23402

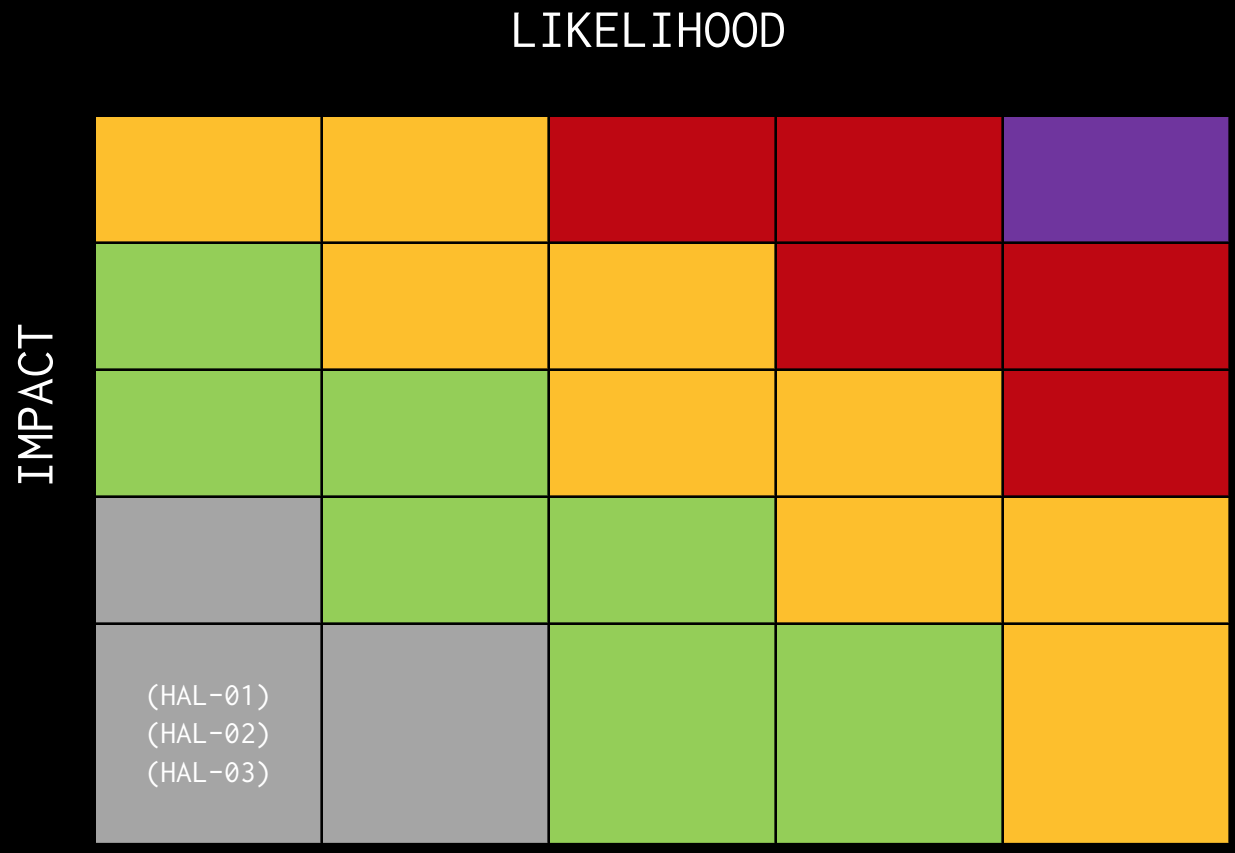
Halborn performed an additional audit of code refactoring and new features included in 0.2.0 version:

- Commit ID: a924a380ac4da82abbb3704cf4fc47b987c0f69d

Out-of-scope:
- third-party libraries and dependencies

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) HARDCODED API KEY	Informational	SOLVED - 04/11/2023
(HAL-02) MISSING AUTHORIZATION FOR GRPC SERVICE SUBSCRIPTION	Informational	ACKNOWLEDGED
(HAL-03) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	SOLVED - 04/11/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) API KEY EXPOSED IN THE CODE - INFORMATIONAL

Description:

API key was found in the examples' folder. The key can be used to authenticate with Helius (Solana infrastructure provider).

Even if the key is inactive (no credits available), security code scanners could flag this issue as a false positive. It's possible that in the future, someone can forget that this key was exposed and add more credits, then all users with access to the repository can use the private RPC endpoint.

Code Location:

Listing 1: examples/send_client.rs (Line 57)

```
56 let solana_rpc_client = Arc::new(RpcClient::new(
57     "https://rpc.helius.xyz/?api-key=46[REDACTED]".to_owned(),
58 ));
59 let last_nonce = Arc::new(AtomicU64::new(
60     get_nonce_master(&solana_rpc_client).await.unwrap(),
61 ));
```

Recommendation:

It is recommended to remove the API key from the code base.

Remediation Plan:

SOLVED: The DeBridge team solved this issue in commit (90b9838ba2c8ef30bfa90deb57c72ff5a5bb0a01): the API key was revoked and removed from the codebase.

3.2 (HAL-02) MISSING AUTHORIZATION FOR GRPC SERVICE SUBSCRIPTION - INFORMATIONAL

Description:

GRPC Service serves events from DeBridge Solana programs stored in the database. Clients can subscribe to GRPC Service using the gRPC protocol and protobuf defined by DeBridge and receive a feed of events.

It was observed that no authorization, access control, and rate-limiting mechanism was implemented in the GRPC Service code. Lack of these protection mechanisms could expose GRPC Service to DOS/DDOS (Denial of Service/Distributed Denial of Service) attacks.

Proof of concept:

Sample client with many subscriptions initialized:

Listing 2: debridge-solana-events-reader/examples/stress_test.rs (Line 17)

```
17 #[tokio::main]
18 async fn main() {
19     let config = Configuration::builder().env().load().unwrap();
20     tracing::info!("Run with configuration: {config:?}");
21
22     let mut client = DebridgeSolanaEventsClient::connect(config.
↳ execute_api_address)
23         .await
24         .expect("Failed to connect server");
25
26     let mut tasks = vec![];
27
28     for subscription in 1..1000 {
29         let subscription = [
30             Subscription {
31                 block_number: 1,
```

```
32                  is_external_call_needed: false,
33              },
34              ...
35              Subscription {
36                  block_number: 0,
37                  is_external_call_needed: false,
38              },
39          ];
40          let request = Request::new(stream::iter(subscription.clone
↳ ()));
41          let mut result = client.get_bridge_created_events(request)
↳ .await.unwrap();
42          let task =
43              tokio::spawn(async move {
44                  result
45                      .into_inner()
46                      .for_each(|event| async move {
47                          if let Ok(e) = event {
48                              match e.bridge_create_event_message.
↳ unwrap() {
49                          bridge_created_event_message::
↳ BridgeCreateEventMessage::Event(event) => {
50                              println!(
51                                  "Created new bridge with token
↳ name: {:?}, native chain id: {:?}",
52                                  event
53                                      .deploy_info
54                                      .as_ref()
55                                      .unwrap()
56                                      .native_token_metadata
57                                      .as_ref()
58                                      .unwrap()
59                                      .name,
60                                  event.deploy_info.as_ref().unwrap
↳ ().native_chain_id,
61                              );
62                          }
63                          bridge_created_event_message::
↳ BridgeCreateEventMessage::Heartbeat(
64                              heartbeat,
65                          ) => {
66                              println!("New heartbeat: {heartbeat:?}
↳ ",)
67                          }
```

```
68                      }
69                          }
70                      })
71                      .await;
72              });
73          tasks.push(task);
74      }
75
76      for task in tasks {
77          task.await.unwrap();
78      }
79  }
```

The database is overloaded with requests:

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

To protect the application against DoS and DDOS attacks, it's recommended to implement access control and rate-limiting mechanisms on the infrastructure level or deploy GRPC Service only in closed/internal networks/subnets.

Remediation Plan:

ACKNOWLEDGED: The DeBridge team acknowledged this finding. The GRPC Service is not intended to be used outside the internal network, so there are no access control and load-tracking mechanisms. The service will be deployed within the DeBridge owned network inside docker/k8s with appropriate network security rules.

3.3 (HAL-03) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL

Description:

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

Code Location:

Listing 3: src/consumer.rs (Line 281)

```
276 let send_event = SendEventBuilder {
277     transaction_hash: tx_signature,
278     slot: slot_number,
279     block_time,
280     block_hash: None,
281     denominator: Denominator::new(transferred.denominator).unwrap
 ↳ (),
282     original_event: transferred,
283     bridge_balance_changed,
284     commitment_level,
285     external_call_data,
286     tracked_by_reader_timestamp: chrono::Utc::now().timestamp() as
 ↳  u64,
287 }
```

Listing 4: src/consumer.rs (Line 295)

```
295 events_storage::insert_send_event(&mut db.acquire().await.unwrap()
 ↳ , send_event)
296     .await
```

```
297     .map_err(|err| {
298         solana_events_parser::transaction_parser::Error::
 ↳ ErrorWhileConsume(format!(
299             "Error while store send event in db: {err:?}",
300         ))
301     })?;
```

Listing 5: src/events_storage.rs (Line 476)

```
469 send_event_message: Some(send_event_message::SendEventMessage::
 ↳ Heartbeat(
470     Heartbeat {
471         last_event_block: hb.last_event_block.unwrap_or_default().
 ↳ to_u64().unwrap(),
472         resync_last_block: hb
473             .resync_last_block
474             .unwrap_or_default()
475             .to_u64()
476             .unwrap(),
477         rpc_last_block: hb.rpc_last_block.unwrap_or_default().
 ↳ to_u64().unwrap(),
478     },
```

Listing 6: src/events_storage.rs (Line 511)

```
502 sender
503     .send(Self {
504         bridge_create_event_message: Some(
505             bridge_created_event_message::BridgeCreateEventMessage
 ↳ ::Heartbeat(Heartbeat {
506                 last_event_block: hb.last_event_block.
 ↳ unwrap_or_default().to_u64().unwrap(),
507                 resync_last_block: hb
508                     .resync_last_block
509                     .unwrap_or_default()
510                     .to_u64()
511                     .unwrap(),
512                 rpc_last_block: hb.rpc_last_block.
 ↳ unwrap_or_default().to_u64().unwrap(),
513             }),
514         ),
515     })
516     .map_err(Box::new)?;
```

Listing 7: src/events_storage.rs (Line 546)

```
537 sender
538             .send(Self {
539                 claim_event_message: Some(claim_event_message::
 ↳ ClaimEventMessage::Heartbeat(
540                     Heartbeat {
541                         last_event_block: hb.last_event_block.
 ↳ unwrap_or_default().to_u64().unwrap(),
542                         resync_last_block: hb
543                             .resync_last_block
544                             .unwrap_or_default()
545                             .to_u64()
546                             .unwrap(),
547                         rpc_last_block: hb.rpc_last_block.
 ↳ unwrap_or_default().to_u64().unwrap(),
548                     },
549                 )),
550             })
551             .map_err(Box::new)?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to use the unwrap function in the production environment because its use can cause panic! and may crash the program without providing verbose error messages. Crashing the system can result in a loss of availability and, in some cases, even the exposure of 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:

SOLVED: The DeBridge team solved this issue in commit (375af243021424afa0546c64f145c4783c39f632): all identified unsafe unwrap functions were replaced by a safer error handling.

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 for which Halborn was testing the program.

4.1 TRANSACTION EVENTS PARSING

Description:

The Events Reader service connects to the Solana node via RPC and WebSocket connection to parse instructions events and store them in a SQL database. The following scenarios related to the Events Reader were tested:

- Only transactions from the predefined program are processed and stored. There is no possibility of polluting DeBridge transaction events with events coming from different Solana programs,
- Only messages from expected program instructions are parsed and stored,
- Transactions events are parsed correctly,
- Transaction events synchronization is working as expected,
- All expected events from transactions are parsed and stored in the database.

Results:

No code vulnerabilities were identified.

Environment for tests
Tests for the Events Reader service were performed using local postgresql instance and Solana test-validator with DeBridge Solana programs (debridge and settings). The environment was configured as follows:

1. Genesis setup

Listing 8: genesis-init.rs (Line 50)

```
50 #[tokio::test]
51 async fn genesis_init() {
52     let mut genesis = solana_test_framework::TestValidatorGenesis
↳ ::default();
53
54     let ledger_path = PathBuf::from(REDACTED);
55     genesis.ledger_path(ledger_path.clone());
56     const SOLANA_CHAIN_ID_RAW: u64 = 7565164;
57     const SOLANA_CHAIN_ID: U256 = U256::from_u64(
↳ SOLANA_CHAIN_ID_RAW);
58     const ETH_CHAIN_ID: U256 = U256::ONE;
59
60     let deploy_info = debridge_settings_program::DeployInfo {
61         chain_id: ETH_CHAIN_ID.to_bytes(),
62         native_token_address: native_mint::ID.try_to_vec().unwrap
↳ (),
63         name: "Wrapped Solana".to_string(),
64         symbol: "SOL".to_string(),
65         decimals: 9,
66     };
67
68     genesis.add_program("debridge_program", debridge_program::ID);
69     genesis.add_program("debridge_settings_program",
↳ debridge_settings_program::ID);
70     genesis.add_program(
71         "mpl_token_metadata",
72         mpl_token_metadata::ID,
73     );
74
75     let protocol_authority = Keypair::from_bytes(REDACTED).unwrap
↳ ();
76     let stop_tap = Keypair::from_bytes(REDACTED).unwrap();
77     let fee_beneficiary = Keypair::from_bytes(REDACTED).unwrap();
78
79     let native_token_mint = spl_token::native_mint::ID;
80
81     let (token_mint, bump_token_mint) = Pubkey::
↳ find_program_address(
82         &[
83             b"WRAPPER_TOKEN",
84             &debridge_settings_program::Bridge::get_bridge_id(
85                 &deploy_info.chain_id,
86                 &deploy_info.native_token_address,
```

```
87              ),
88          ],
89          &debridge_settings_program::ID,
90      );
91      let (bridge_data, bump_bridge_data) = Pubkey::
↳ find_program_address(
92          &[b"BRIDGE", token_mint.as_ref()],
93          &debridge_settings_program::ID,
94      );
95      let (chain_support_info, bump_bridge_data) = Pubkey::
↳ find_program_address(
96          &[b"CHAIN_SUPPORT_INFO", &ETH_CHAIN_ID.to_bytes()],
97          &debridge_settings_program::ID,
98      );
99      let (mint_authority, bump_mint_authority) = Pubkey::
↳ find_program_address(
100         &[b"BRIDGE_MINTER", bridge_data.as_ref()],
101         &debridge_program::ID,
102     );
103
104     let (token_metadata, bump_token_metadata) = Pubkey::
↳ find_program_address(
105         &[
106             b"metadata",
107             mpl_token_metadata::ID.as_ref(),
108             &token_mint.as_ref(),
109         ],
110         &mpl_token_metadata::ID,
111     );
112
113     let staking_wallet = get_associated_token_address(&
↳ mint_authority, &token_mint);
114
115     genesis.add_account_with_lamports(
116         protocol_authority.pubkey(),
117         system_program::ID,
118         10_000_000_000_000_000,
119     );
120
121     genesis.add_account_with_lamports(
122         stop_tap.pubkey(),
123         system_program::ID,
124         10_000_000_000_000_000,
125     );
```

```
126
127     genesis.add_account_with_lamports(
128         fee_beneficiary.pubkey(),
129         system_program::ID,
130         10_000_000_000_000_000,
131     );
132
133     genesis.add_token_mint(native_token_mint, None, 1, 0, None);
134     genesis.add_token_mint(token_mint, Some(mint_authority), 10
 ↳ _000_000_000, 0, None); //comment this line if using mint bridge
135     genesis.add_token_account(
136         staking_wallet,
137         token_mint,
138         mint_authority,
139         10_000_000_000,
140         None,
141         None,
142         0,
143         None,
144     );
145
146     let sender = Keypair::from_bytes(&[REDACTED]).unwrap();
147     let sender_wallet = Pubkey::from_str("8
 ↳ rcFeCFmy5aK1EroGCmcLyGmgfFXL7vPdZqhobbirGpo").unwrap();
148     genesis.add_token_account(
149         sender_wallet,
150         token_mint,
151         sender.pubkey(),
152         10,
153         None,
154         None,
155         0,
156         None,
157     );
158     genesis.add_account_with_lamports(sender.pubkey(),
 ↳ system_program::ID, 10_000_000_000_000_000);
159     let (test_validator, validator_kp) = genesis.start_async().
 ↳ await;
160 }
```

2. Initialize programs and settings

Listing 9: init.rs (Line 52)

```
52 #[tokio::test(flavor = "multi_thread", worker_threads = 1)]
53 async fn init() {
54     let url = "http://localhost:8899".to_string();
55     let mut rpc_client = RpcClient::new(url);
56 
57     let (state_settings, bump_state_setting) =
58         Pubkey::find_program_address(&[b"DEBRIDGE_STATE"], &
↳ debridge_settings_program::ID);
59 
60     let mut ix = Instruction::new_with_bytes(
61         debridge_settings_program::ID,
62         debridge_settings_program::instruction::InitializeState {
63             confirmation_threshold: 8,
64             excess_confirmations: 3,
65             min_confirmations: 2,
66         }
67         .data()
68         .as_ref(),
69         debridge_settings_program::accounts::InitializeState {
70             state: state_settings,
71             protocol_authority: protocol_authority.pubkey(),
72             stop_tap: stop_tap.pubkey(),
73             fee_beneficiary: fee_beneficiary.pubkey(),
74             system_program: system_program::ID,
75             settings_program: debridge_settings_program::ID,
76             payer: protocol_authority.pubkey(),
77         }
78         .to_account_metas(Some(false)),
79     );
80 
81     let debridge_settings_program_json_file = "REDACTED/settings.
↳ json";
82 
83     let keypair_bytes =
84         solana_sdk::signer::keypair::read_keypair_file(
↳ debridge_settings_program_json_file)
85             .expect("Can't read debridge_settings_program-keypair.
↳ json");
86     let mut tx = rpc_client
87         .transaction_from_instructions(
88             &[ix],
89             &protocol_authority,
90             vec![&protocol_authority, &keypair_bytes],
```

```
91          )
92          .await
93          .unwrap();
94
95      let mut ix = Instruction::new_with_bytes(
96          debridge_settings_program::ID,
97          debridge_settings_program::instruction::AddOracle {
98              oracle: required_oracle,
99              is_required: true,
100         }
101         .data()
102         .as_ref(),
103         debridge_settings_program::accounts::UpdateState {
104             state: state_settings,
105             protocol_authority: protocol_authority.pubkey(),
106         }
107         .to_account_metas(Some(false)),
108     );
109
110     let mut tx = rpc_client
111         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
112         .await
113         .unwrap();
114
115     rpc_client.send_and_confirm_transaction(&tx).unwrap();
116
117     let mut ix = Instruction::new_with_bytes(
118         debridge_settings_program::ID,
119         debridge_settings_program::instruction::AddOracle {
120             oracle: oracle,
121             is_required: false,
122         }
123         .data()
124         .as_ref(),
125         debridge_settings_program::accounts::UpdateState {
126             state: state_settings,
127             protocol_authority: protocol_authority.pubkey(),
128         }
129         .to_account_metas(Some(false)),
130     );
131
132     let mut tx = rpc_client
133         .transaction_from_instructions(&[ix], &protocol_authority,
```

```
 ↳  vec![&protocol_authority])
134         .await
135         .unwrap();
136
137     rpc_client.send_and_confirm_transaction(&tx).unwrap();
138
139     let mut ix = Instruction::new_with_bytes(
140         debridge_settings_program::ID,
141         debridge_settings_program::instruction::
 ↳ UpdateMinConfirmationCount {
142             min_confirmation: 2,
143         }
144         .data()
145         .as_ref(),
146         debridge_settings_program::accounts::UpdateState {
147             state: state_settings,
148             protocol_authority: protocol_authority.pubkey(),
149         }
150         .to_account_metas(Some(false)),
151     );
152
153     let mut tx = rpc_client
154         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
155         .await
156         .unwrap();
157
158     rpc_client.send_and_confirm_transaction(&tx).unwrap();
159
160     let mut ix = Instruction::new_with_bytes(
161         debridge_settings_program::ID,
162         debridge_settings_program::instruction::
 ↳ UpdateExcessConfirmationTimeslot {
163             new_excess_confirmation_timeslot_seconds: 3,
164         }
165         .data()
166         .as_ref(),
167         debridge_settings_program::accounts::UpdateState {
168             state: state_settings,
169             protocol_authority: protocol_authority.pubkey(),
170         }
171         .to_account_metas(Some(false)),
172     );
173
```

```
174     let mut tx = rpc_client
175         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
176         .await
177         .unwrap();
178 
179     rpc_client.send_and_confirm_transaction(&tx).unwrap();
180 
181     let mut ix = Instruction::new_with_bytes(
182         debridge_settings_program::ID,
183         debridge_settings_program::instruction::
 ↳ UpdateExcessConfirmationCount {
184             excess_confirmations: 3,
185         }
186         .data()
187         .as_ref(),
188         debridge_settings_program::accounts::UpdateState {
189             state: state_settings,
190             protocol_authority: protocol_authority.pubkey(),
191         }
192         .to_account_metas(Some(false)),
193     );
194 
195     let mut tx = rpc_client
196         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
197         .await
198         .unwrap();
199 
200     rpc_client.send_and_confirm_transaction(&tx).unwrap();
201 
202     let mut ix = Instruction::new_with_bytes(
203         debridge_settings_program::ID,
204         debridge_settings_program::instruction::
 ↳ UpdateConfirmationThreshold {
205             confirmation_threshold: 1,
206         }
207         .data()
208         .as_ref(),
209         debridge_settings_program::accounts::UpdateState {
210             state: state_settings,
211             protocol_authority: protocol_authority.pubkey(),
212         }
213         .to_account_metas(Some(false)),
```

```
214     );
215
216     let mut tx = rpc_client
217         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
218         .await
219         .unwrap();
220
221     rpc_client.send_and_confirm_transaction(&tx).unwrap();
222
223     let mut ix = Instruction::new_with_bytes(
224         debridge_settings_program::ID,
225         debridge_settings_program::instruction::UpdateGlobalFee {
226             global_fixed_fee: 100,        //aka 1%
227             global_transfer_fee_bps: 100, // aka 1%
228         }
229         .data()
230         .as_ref(),
231         debridge_settings_program::accounts::UpdateState {
232             state: state_settings,
233             protocol_authority: protocol_authority.pubkey(),
234         }
235         .to_account_metas(Some(false)),
236     );
237
238     let mut tx = rpc_client
239         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
240         .await
241         .unwrap();
242
243     rpc_client.send_and_confirm_transaction(&tx).unwrap();
244
245     let mut ix = Instruction::new_with_bytes(
246         debridge_settings_program::ID,
247         debridge_settings_program::instruction::
 ↳ UpdateFeeBeneficiary {}
248             .data()
249             .as_ref(),
250         debridge_settings_program::accounts::
 ↳ UpdateStateWithAccount {
251             update_state: debridge_settings_program::accounts::
 ↳ UpdateState {
252                 state: state_settings,
```

```
253                 protocol_authority: protocol_authority.pubkey(),
254             },
255             account: fee_beneficiary.pubkey(),
256         }
257         .to_account_metas(Some(false)),
258     );
259
260     let mut tx = rpc_client
261         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
262         .await
263         .unwrap();
264
265     rpc_client.send_and_confirm_transaction(&tx).unwrap();
266
267     let mut ix = Instruction::new_with_bytes(
268         debridge_settings_program::ID,
269         debridge_settings_program::instruction::
 ↳ UpdateProtocolAuthority {}
270             .data()
271             .as_ref(),
272         debridge_settings_program::accounts::
 ↳ UpdateProtocolAuthority {
273             state: state_settings,
274             protocol_authority: protocol_authority.pubkey(),
275             new_protocol_authority: protocol_authority.pubkey(),
276         }
277         .to_account_metas(Some(false)),
278     );
279
280     let mut tx = rpc_client
281         .transaction_from_instructions(
282             &[ix],
283             &protocol_authority,
284             vec![&protocol_authority, &protocol_authority],
285         )
286         .await
287         .unwrap();
288
289     rpc_client.send_and_confirm_transaction(&tx).unwrap();
290
291     let mut ix = Instruction::new_with_bytes(
292         debridge_settings_program::ID,
293         debridge_settings_program::instruction::
```

```
 ↳ UpdateChainSupportInfo {
294             target_chain_id: ETH_CHAIN_ID.to_bytes(),
295             is_supported: true,
296             fixed_fee: None,
297             transfer_fee: None,
298             chain_address_len: 1,
299         }
300         .data()
301         .as_ref(),
302         debridge_settings_program::accounts::
 ↳ UpdateChainSupportInfo {
303             state: state_settings,
304             protocol_authority: protocol_authority.pubkey(),
305             payer: protocol_authority.pubkey(),
306             chain_support_info,
307             system_program: system_program::ID,
308         }
309         .to_account_metas(Some(false)),
310     );
311
312     let mut tx = rpc_client
313         .transaction_from_instructions(
314             &[ix],
315             &protocol_authority,
316             vec![&protocol_authority, &protocol_authority],
317         )
318         .await
319         .unwrap();
320
321     rpc_client.send_and_confirm_transaction(&tx).unwrap();
322
323     let bridge_id = deploy_info.get_bridge_id().unwrap();
324     let deploy_id = deploy_info
325         .calculate_id(Some(bridge_id.clone()))
326         .unwrap()
327         .to_bytes();
328
329     let mut msg = deploy_id.clone().try_to_vec().unwrap();
330
331     let signatures = test_oracles.sign(&msg);
332
333     let msg_text = [
334         format!("\x19Ethereum Signed Message:\n{}", msg.len()).
 ↳ into_bytes(),
```

```
335         msg.clone(),
336     ]
337     .concat();
338 
339     let ix1 =
340         Instruction::new_secp256k1_instruction(msg_text.as_slice()
 ↳ , 0, signatures.into_iter())
341             .unwrap();
342 
343     let (confirmation_storage, bump_confirmation_storage) =
344         Pubkey::find_program_address(&[b"SIGNATURES", &msg], &
 ↳ debridge_settings_program::ID);
345 
346     let mut ix2 = Instruction::new_with_bytes(
347         debridge_settings_program::ID,
348         debridge_settings_program::instruction::StoreConfirmations
 ↳  { msg: msg }
349             .data()
350             .as_ref(),
351         debridge_settings_program::accounts::
 ↳ UpdateConfirmationStorage {
352             state: state_settings,
353             confirmation_storage: confirmation_storage,
354             instructions: InstructionID,
355             system_program: system_program::ID,
356             payer: protocol_authority.pubkey(),
357         }
358         .to_account_metas(Some(false)),
359     );
360 
361     let mut tx = rpc_client
362         .transaction_from_instructions(&[ix1, ix2], &
 ↳ protocol_authority, vec![&protocol_authority])
363         .await
364         .unwrap();
365 
366     rpc_client.send_and_confirm_transaction(&tx).unwrap();
367 
368     let (token_metadata_master, bump_token_metadata_master) =
369         Pubkey::find_program_address(&[b"TOKEN_METADATA"], &
 ↳ debridge_settings_program::ID);
370 
371     let mut ix = Instruction::new_with_bytes(
372         debridge_settings_program::ID,
```

```
373         debridge_settings_program::instruction::InitMetadataMaster
 ↳ {
374             init_prefix: "test".to_string(),
375             init_postfix: "test".to_string(),
376         }
377         .data()
378         .as_ref(),
379         debridge_settings_program::accounts::InitMetadataMaster {
380             token_metadata_master: token_metadata_master,
381             state: state_settings,
382             protocol_authority: protocol_authority.pubkey(),
383             payer: protocol_authority.pubkey(),
384             system_program: system_program::ID,
385         }
386         .to_account_metas(Some(false)),
387     );
388
389     let mut tx = rpc_client
390         .transaction_from_instructions(
391             &[ix],
392             &protocol_authority,
393             vec![&protocol_authority, &protocol_authority],
394         )
395         .await
396         .unwrap();
397
398     rpc_client.send_and_confirm_transaction(&tx).unwrap();
399
400     let mut ix = Instruction::new_with_bytes(
401         debridge_settings_program::ID,
402         debridge_settings_program::instruction::
 ↳ UpdateMetadataMaster {
403             new_prefix: "test".to_string(),
404             new_postfix: "test".to_string(),
405         }
406         .data()
407         .as_ref(),
408         debridge_settings_program::accounts::UpdateMetadataMaster
 ↳ {
409             token_metadata_master: token_metadata_master,
410             state: state_settings,
411             protocol_authority: protocol_authority.pubkey(),
412         }
413         .to_account_metas(Some(false)),
```

```
414     );
415
416     let mut tx = rpc_client
417         .transaction_from_instructions(&[ix], &protocol_authority,
 ↳  vec![&protocol_authority])
418         .await
419         .unwrap();
420
421     rpc_client.send_and_confirm_transaction(&tx).unwrap();
422 }
```

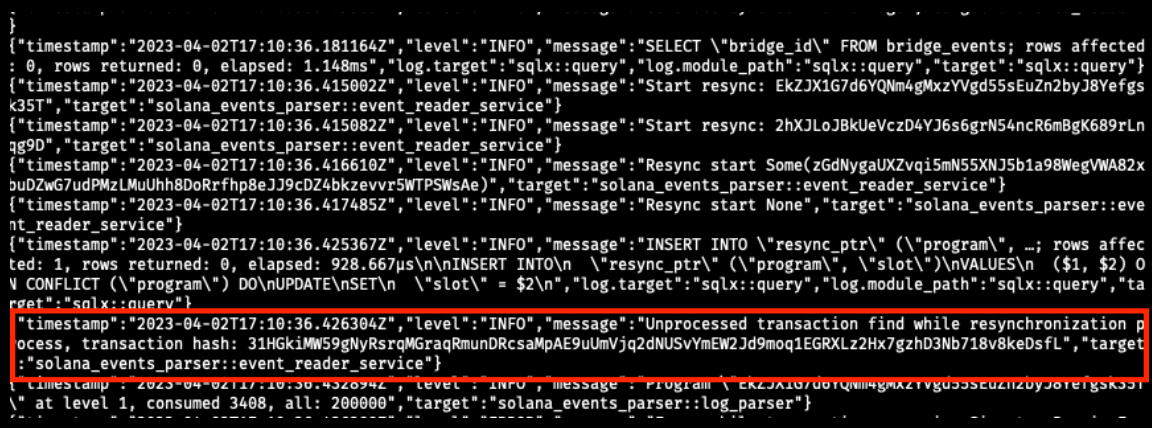
Events synchronization tests

Sending new MintBridgeCreated event:

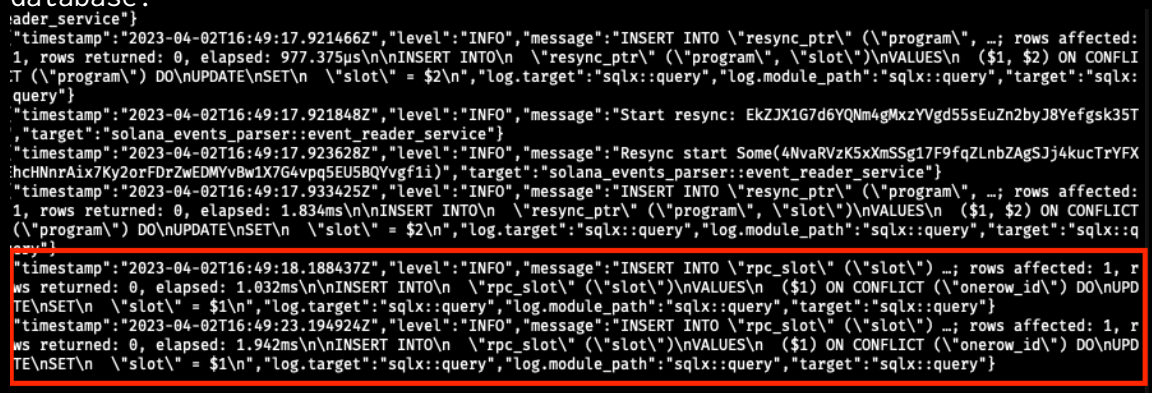
```
Transaction hash: 31HGkiMW59gNyRsrqMGraqRmunDRcsaMpAE9uUmVjq2dNUSvYmEW2Jd9
moq1EGRXLz2Hx7gzhD3Nb718v8keDsfL
{
  data: {
    bridgeId: [
      162, 132, 189,  45, 119,  78,  19,
      196, 195, 132,  17, 209, 155,  83,
      119,  80,  96, 197, 142, 139, 212,
      245, 218, 171, 152, 140, 169,  51,
      183, 149,  52, 185
    ],
    deployId: [
      0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 1
    ],
    wrapperTokenAddress: [
      0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 2
    ]
  },
  name: 'MintBridgeCreated'
}
➜  node-app
```

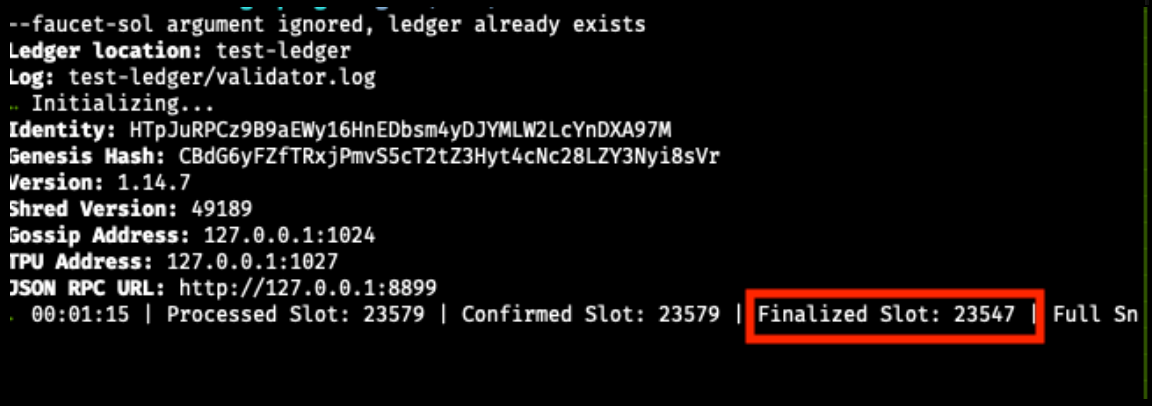
A new transaction was found and synchronized:

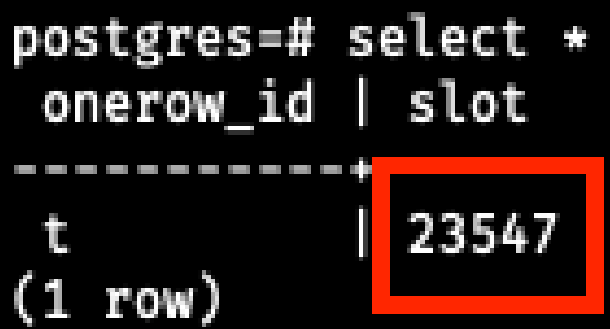
MANUAL TESTING



Current finalized slot is correctly pulled from chain and kept in the database:







Transactions are synchronized correctly even with broken WebSocket connection

In this scenario, wrong URL to WebSocket endpoint was provided to simulate failure in RPC node.

Sending new Send transaction:

Listing 10: solana-contracts/programs/debridge/tests/send.rs (Line 182)

```
182 let url = "http://localhost:8899".to_string();
183 let mut rpc_client = RpcClient::new(url);
184 let mut ix = Instruction::new_with_bytes(
185     debridge_program::ID,
186     debridge_program::instruction::Send {
187         amount: 1,
188         target_chain_id: ETH_CHAIN_ID.to_bytes(),
189         receiver: vec![0x00],
190         is_use_asset_fee: false,
191         submission_params: None,
192         referral_code: None,
193     }
194     .data()
195     .as_ref(),
196     debridge_program::accounts::Sending {
197         nonce_storage,
198         bridge_ctx: debridge_program::accounts::BridgeCtx {
199             bridge: bridge_data,
200             token_mint,
201             staking_wallet,
202             mint_authority,
203             chain_support_info,
204             settings_program: debridge_settings_program::ID,
205             spl_token_program: spl_token::id(),
206         },
207         state_ctx: debridge_program::accounts::StateCtx {
208             state: state_settings,
209             fee_beneficiary: fee_beneficiary.pubkey(),
210         },
211         send_from_wallet: sender_wallet,
212         send_from: sender.pubkey(),
213         system_program: system_program::ID,
214         external_call_storage: Pubkey::new_unique(),
215         external_call_meta: Pubkey::new_unique(),
```

```rust
            discount: no_discount,
            bridge_fee,
        }
        .to_account_metas(Some(false)),
);

let mut tx =  transaction_from_instructions(&mut rpc_client, &[ix
 ↳ ], &sender, vec![&sender])
        .await
        .unwrap();

rpc_client.send_and_confirm_transaction(&tx).unwrap();
```

New transaction in cluster logs:

```
Streaming transaction logs. Confirmed commitment
Transaction executed in slot 5957:
  Signature: on9RpCWHiytcua82YKWM7gEBrT6jugsW4GYVar1o2SJ1CLgXgyCdckgGQ3p8d7WEZXWcfRidyLeV8r24jhPgrtP
  Status: Ok
  Log Messages:
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ invoke [1]
    Program log: Instruction: Send
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: Transfer
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 164021 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]
    Program log: Instruction: StakeNativeFee
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 154379 compute units
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]
    Program log: Instruction: StakeBalance
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 138557 compute units
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success
    Program data: 53J9Oon4lMkGBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAkAEAAAAAAAAAA5AtUAgAAAA==
    Program data: FYTvQJLvpkQAYdSl4jv64rcGVO2YBPnXpricMG85WrtACS5e2QX0KpJoVr2q96Bqe3TAwGKSQZsAVfuehEN+22BrGmy5L8SwHQEAAAAAAAAAAQAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAGQAAAAAAAAAAAAAAAAAADufQJ6uDZwC52Alrtp/8OoNLw1/hkFlQUzUPjXXo22RQAAAAA=
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ consumed 87727 of 200000 compute units
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ success
```

New transaction was found and synchronized in the most recent synchronization event:

```
events_parser::event_reader_service"}
{"timestamp":"2023-04-20T07:37:46.798162Z","level":"INFO","message":"Resync start Some(53H2urgfKXwmDKPuqrVcFD2pNvTLhCHZL4HGNhF9CBh5bx3oc6f6h7gxVUa
TLdzH258t8o4VrJ5p95qEFyQKneB9)","target":"solana_events_parser::event_reader_service"}
{"timestamp":"2023-04-20T07:37:46.800221Z","level":"INFO","message":"Unprocessed transaction find while resynchronization process, transaction has
h: on9RpCWHiytcua82YKWM7gEBrT6jugsW4GYVar1o2SJ1CLgXgyCdckgGQ3p8d7WEZXWcfRidyLeV8r24jhPgrtP","target":"solana_events_parser::event_reader_service"}
{"timestamp":"2023-04-20T07:37:46.800623Z","level":"INFO","message":"Unprocessed transaction find while resynchronization process, transaction has
h: on9RpCWHiytcua82YKWM7gEBrT6jugsW4GYVar1o2SJ1CLgXgyCdckgGQ3p8d7WEZXWcfRidyLeV8r24jhPgrtP","target":"solana_events_parser::event_reader_service"}
{"timestamp":"2023-04-20T07:37:46.816497Z","level":"INFO","message":"Program \"TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA\" at level 2, consumed
4645, all: 164021","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.816958Z","level":"INFO","message":"Program \"FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc\" at level 2, consumed
12443, all: 154379","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.817293Z","level":"INFO","message":"Program \"FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc\" at level 2, consumed
12443, all: 138557","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.817785Z","level":"INFO","message":"Program \"6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ\" at level 1, consumed
87727, all: 200000","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.830508Z","level":"INFO","message":"Transaction on9RpCWHiytcua82YKWM7gEBrT6jugsW4GYVar1o2SJ1CLgXgyCdckgGQ3p8d7WEZ
XWcfRidyLeV8r24jhPgrtP consumed","target":"solana_events_parser::event_reader_service"}
{"timestamp":"2023-04-20T07:37:46.832251Z","level":"INFO","message":"Set last resynced tx to on9RpCWHiytcua82YKWM7gEBrT6jugsW4GYVar1o2SJ1CLgXgyCdc
kgGQ3p8d7WEZXWcfRidyLeV8r24jhPgrtP","target":"solana_events_parser::event_reader_service"}
{"timestamp":"2023-04-20T07:37:46.834721Z","level":"INFO","message":"INSERT INTO \"resync_ptr\" (\"program\", …; rows affected: 1, rows returned:
0, elapsed: 1.298ms\n\nINSERT INTO\n  \"resync_ptr\" (\"program\", \"slot\")\nVALUES\n  ($1, $2) ON CONFLICT (\"program\") DO\nUPDATE\nSET\n  \"sl
ot\" = $2\n","log.target":"sqlx::query","log.module_path":"sqlx::query","target":"sqlx::query"}
{"timestamp":"2023-04-20T07:37:46.846246Z","level":"INFO","message":"Program \"TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA\" at level 2, consumed
4645, all: 164021","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.846706Z","level":"INFO","message":"Program \"FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc\" at level 2, consumed
12443, all: 154379","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.847055Z","level":"INFO","message":"Program \"FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc\" at level 2, consumed
12443, all: 138557","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.847560Z","level":"INFO","message":"Program \"6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ\" at level 1, consumed
87727, all: 200000","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T07:37:46.861516Z","level":"WARN","message":"Error while getting external call data: Error { request: None, kind: RpcError
ForUser(\"AccountNotFound: pubkey=11111110Lbz7JHiBTspS962RLKV8GndWFwiEaqKM\")) }","target":"solana_events::consumer"}
{"timestamp":"2023-04-20T07:37:46.861592Z","level":"INFO","message":"Received amount in transaction on9RpCWHiytcua82YKWM7gEBrT6jugsW4GYVar1o2SJ1CL
gXgyCdckgGQ3p8d7WEZXWcfRidyLeV8r24jhPgrtP is 0x0000000000000000000000000000000000000000000000000000000000000001","log.target":"debridge_protobuf_r
s::debridge_solana_service","log.module_path":"debridge_protobuf_rs::debridge_solana_service","log.file":"/Users/mb/.cargo/git/checkouts/debridge-
protobuf-rs-06d3b571b792a870/1f5ca1b/src/lib.rs","log.line":149,"target":"debridge_protobuf_rs::debridge_solana_service"}
{"timestamp":"2023-04-20T07:37:46.861968Z","level":"INFO","message":"Processing send event: 0x61d4a5e23bfae2b70654ed9804f9d7a6b89c306f395abb40092e
5ed905f42a92","target":"solana_events::events_storage"}
{"timestamp":"2023-04-20T07:37:46.864110Z","level":"INFO","message":"INSERT INTO send_events (slot_number, …; rows affected: 1, rows returned: 0,
elapsed: 550.250µs\n\nINSERT INTO\n  send_events (slot_number, nonce, protobuf_buffer)\nVALUES\n  ($1, $2, $3) ON CONFLICT (nonce) DO\nUPDATE\nSET
\n  \"protobuf_buffer\" = $3\nWHERE\n  send_events.nonce = $2\n","log.target":"sqlx::query","log.module_path":"sqlx::query","target":"sqlx::query"
}
```

Transactions are synchronized correctly after unhealthy node recover

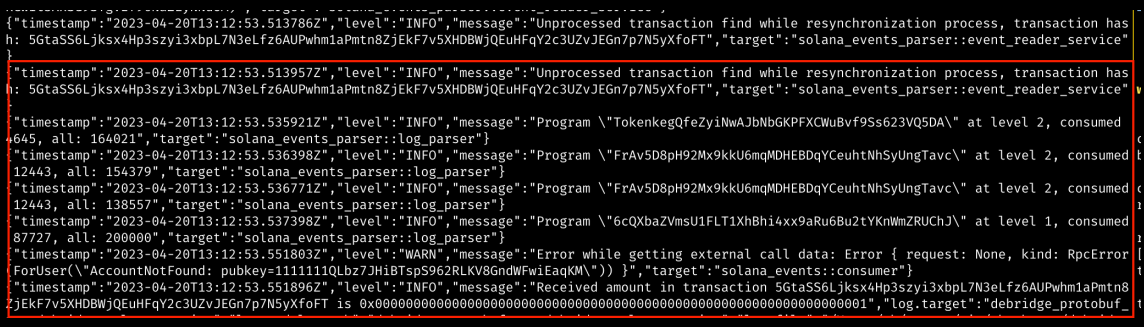
In this scenario, two Solana local nodes were created. The Events Reader service was connected to an unstable node with crash simulation, and transactions were sent to the always healthy node.

The Events Reader service was able to synchronize remaining transactions when the node became healthy again.

A new transaction in cluster logs:

```
Streaming transaction logs. Confirmed commitment
Transaction executed in slot 33091:
  Signature: 5GtaSS6Ljksx4Hp3szyi3xbpL7N3eLfz6AUPwhm1aPmtn8ZjEkF7v5XHDBWjQEuHFqY2c3UZvJEGn7p7N5yXfoFT
  Status: Ok
  Log Messages:
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ invoke [1]
    Program log: Instruction: Send
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: Transfer
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 164021 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]
    Program log: Instruction: StakeNativeFee
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 154379 compute units
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]
    Program log: Instruction: StakeBalance
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 138557 compute units
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success
    Program data: 53J9Oon4lMkGCQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAhAMAAAAAAAAAA5AtUAgAAAA==
    Program data: FYTvQJLvpkQA8HD2l94g5EJF14Tja3m0UkQ0UB4WA6K8oQKSQou5DRVoVr2q96Bqe3TAwGKSQZsAVfuehEN+22BrGmy5L8SwHQEAAAAAAAAAAQAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAGQAAAAAAAAAAAAAAAAAADufQJ6uDZwC52Alrtp/8OoNLw1/hkFlQUzUPjXXo22RQAAAAA=
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ consumed 87727 of 200000 compute units
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ success
```

A new transaction was found and synchronized:



Transactions with errors are not parsed

Sending transaction composed with correct and incorrect instructions:

Listing 11: solana-contracts/programs/debridge/tests/send-with-err.rs (Line 185)

```
185 let url = "http://localhost:8899".to_string();
186 let mut rpc_client = RpcClient::new(url);
187
188 // correct IX
189 let mut ix1 = Instruction::new_with_bytes(
190     debridge_program::ID,
191     debridge_program::instruction::Send {
192         amount: 1,
193         target_chain_id: ETH_CHAIN_ID.to_bytes(),
194         receiver: vec![0x00],
195         is_use_asset_fee: false,
196         submission_params: None,
197         referral_code: None,
198     }
199     .data()
200     .as_ref(),
201     debridge_program::accounts::Sending {
202         nonce_storage,
203         bridge_ctx: debridge_program::accounts::BridgeCtx {
204             bridge: bridge_data,
205             token_mint,
206             staking_wallet,
207             mint_authority,
208             chain_support_info,
209             settings_program: debridge_settings_program::ID,
210             spl_token_program: spl_token::id(),
211         },
212         state_ctx: debridge_program::accounts::StateCtx {
213             state: state_settings,
```

```
214             fee_beneficiary: fee_beneficiary.pubkey(),
215         },
216         send_from_wallet: sender_wallet,
217         send_from: sender.pubkey(),
218         system_program: system_program::ID,
219         external_call_storage: Pubkey::new_unique(),
220         external_call_meta: Pubkey::new_unique(),
221         discount: no_discount,
222         bridge_fee,
223     }
224     .to_account_metas(Some(false)),
225 );
226 
227 // incorrect IX
228 let mut ix2 = Instruction::new_with_bytes(
229     debridge_program::ID,
230     debridge_program::instruction::Send {
231         amount: 1,
232         target_chain_id: ETH_CHAIN_ID.to_bytes(),
233         receiver: vec![0x00],
234         is_use_asset_fee: false,
235         submission_params: None,
236         referral_code: None,
237     }
238     .data()
239     .as_ref(),
240     debridge_program::accounts::Sending {
241         nonce_storage,
242         bridge_ctx: debridge_program::accounts::BridgeCtx {
243             bridge: bridge_data,
244             token_mint,
245             staking_wallet,
246             mint_authority,
247             chain_support_info,
248             settings_program: debridge_settings_program::ID,
249             spl_token_program: spl_token::id(),
250         },
251         state_ctx: debridge_program::accounts::StateCtx {
252             state: state_settings,
253             fee_beneficiary: fee_beneficiary.pubkey(),
254         },
255         send_from_wallet: sender2_wallet,
256         send_from: sender2.pubkey(), // incorrect sender should
 ↳ fail
```

```
257         system_program: system_program::ID,
258         external_call_storage: Pubkey::new_unique(),
259         external_call_meta: Pubkey::new_unique(),
260         discount: no_discount,
261         bridge_fee,
262     }
263     .to_account_metas(Some(false)),
264 );
265
266 // correct ix
267 let mut ix3 = Instruction::new_with_bytes(
268     debridge_program::ID,
269     debridge_program::instruction::Send {
270         amount: 1,
271         target_chain_id: ETH_CHAIN_ID.to_bytes(),
272         receiver: vec![0x00],
273         is_use_asset_fee: false,
274         submission_params: None,
275         referral_code: None,
276     }
277     .data()
278     .as_ref(),
279     debridge_program::accounts::Sending {
280         nonce_storage,
281         bridge_ctx: debridge_program::accounts::BridgeCtx {
282             bridge: bridge_data,
283             token_mint,
284             staking_wallet,
285             mint_authority,
286             chain_support_info,
287             settings_program: debridge_settings_program::ID,
288             spl_token_program: spl_token::id(),
289         },
290         state_ctx: debridge_program::accounts::StateCtx {
291             state: state_settings,
292             fee_beneficiary: fee_beneficiary.pubkey(),
293         },
294         send_from_wallet: sender_wallet,
295         send_from: sender.pubkey(),
296         system_program: system_program::ID,
297         external_call_storage: Pubkey::new_unique(),
298         external_call_meta: Pubkey::new_unique(),
299         discount: no_discount,
300         bridge_fee,
```

```
    }
    .to_account_metas(Some(false)),
);

// tx only signed with one sender
let mut tx =  transaction_from_instructions(&mut rpc_client, &[ix1
↳ ,ix2,ix3], &sender, vec![&sender, &sender2])
    .await
    .unwrap();

rpc_client.send_and_confirm_transaction(&tx).unwrap();
```

Transaction was reverted and correctly ignored by the Events Reader service. Transaction error:

```
    Finished test [unoptimized + debuginfo] target(s) in 0.78s
     Running tests/send-with-err.rs (/Users/mb/projects/hal_projects/debridge/solana-contracts/target/debug/deps/send_with_err-a30ce95a9a7110ab)
thread 'halborn_update_after_claim' panicked at 'called `Result::unwrap()` on an `Err` value: Error { request: Some(SendTransaction), kind: RpcEr
ror(RpcResponseError { code: -32002, message: "Transaction simulation failed: Error processing Instruction 1: custom program error: 0x7d3", data:
 SendTransactionPreflightFailure(RpcSimulateTransactionResult { err: Some(InstructionError(1, Custom(2003))), logs: Some(["Program 6cQXbaZVmsU1FL
T1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ invoke [1]", "Program log: Instruction: Send", "Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]", "
Program log: Instruction: Transfer", "Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 564021 compute units", "Program Tokenk
egQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success", "Program 11111111111111111111111111111111 invoke [2]", "Program 11111111111111111111111111111111
success", "Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]", "Program log: Instruction: StakeNativeFee", "Program FrAv5D8pH92Mx9k
kU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 554379 compute units", "Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success", "Program F
rAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]", "Program log: Instruction: StakeBalance", "Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSy
UngTavc consumed 12443 of 538557 compute units", "Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success", "Program data: 53J9Oon4lMkGBgAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAWAIAAAAAAAAA5AtUAgAAAA==", "Program data: FYTvQJLvpkQAnOL72vP0Zbo5BU1kMFQ6ub4JhxSoUlS9DzPJb5R8rPRoVr2q96Bqe3TAwGKSQZsA
VfuehEN+22BrGmy5L8SwHQEAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAGQAAAAAAAAAAAAAAAAAA
ADufQJ6uDZwC52Alrtp/8OoNLw1/hkFlQUzUPjXXo22RQAAAAA=", "Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ consumed 87727 of 600000 compute unit
s", "Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ success", "Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ invoke [1]", "Program l
og: Instruction: Send", "Program log: AnchorError caused by account: send_from_wallet. Error Code: ConstraintRaw. Error Number: 2003. Error Messa
ge: A raw constraint was violated.", "Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ consumed 30206 of 512273 compute units", "Program 6cQX
baZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ failed: custom program error: 0x7d3"]), accounts: None, units_consumed: Some(87727), return_data: None }
) }) }', programs/debridge/tests/send-with-err.rs:309:50
```

No new entries were created for Send event:

```
postgres=# SELECT * from send_events;
postgres=# SELECT nonce,slot_number from send_events;
 nonce | slot_number
-------+-------------
     0 |        2145
     1 |        3298
     2 |        4007
     3 |        4193
     4 |        7656
(5 rows)
```

Feeding service with fake transactions

In this scenario, fake DeBridge settings and debridge programs were deployed in the local cluster:

Listing 12: security-tests/src/for-report.rs (Line 56)

```
56 async fn async_main() {
57     let debridge_program = Pubkey::from_str("6
↳ cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ").unwrap();
58     let debridge_settings = Pubkey::from_str("
↳ FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc").unwrap();
59     let debridge_program_malicious = Pubkey::from_str("
↳ CjVLsQyCFNpgfMLQ9F5iobzxScWL2FYx3r4sGiHdEJub").unwrap();
60     let debridge_settings_malicious = Pubkey::from_str("
↳ yoUBkA9CyCUQHQNT55KGwbnDEVPRXzQAkfdn5r3xRvj").unwrap();
61     let native_token_mint = Pubkey::from_str("
↳ So11111111111111111111111111111111111111112").unwrap();
62     let mut genesis = solana_test_framework::TestValidatorGenesis
↳ ::default();
63
64     let ledger_path = PathBuf::from(
65         "REDACTED",
66     );
67     std::fs::remove_dir_all(ledger_path.clone()).unwrap();
68     genesis.ledger_path(ledger_path.clone());
69     const SOLANA_CHAIN_ID_RAW: u64 = 7565164;
70     const SOLANA_CHAIN_ID: U256 = U256::from_u64(
↳ SOLANA_CHAIN_ID_RAW);
71     const ETH_CHAIN_ID: U256 = U256::ONE;
72
73     let deploy_info = DeployInfo {
74         chain_id: ETH_CHAIN_ID.to_bytes(),
75         native_token_address: native_token_mint.try_to_vec().
↳ unwrap(),
76         name: "Wrapped Solana".to_string(),
77         symbol: "SOL".to_string(),
78         decimals: 9,
79     };
80
81     // Add malicious programs
82     genesis.add_program("REDACTED", debridge_program_malicious);
83     println!("debridge program id: {:?}",
↳ debridge_program_malicious);
84
85     genesis.add_program("REDACTED", debridge_settings_malicious);
86     println!(
87         "debridge settings program id: {:?}",
88         debridge_settings_malicious
89     );
```

```
90
91      // Add valid programs
92      genesis.add_program("REDACTED", debridge_program);
93      println!("debridge program id: {:?}", debridge_program);
94
95      genesis.add_program("REDACTED", debridge_settings);
96      println!(
97          "debridge settings program id: {:?}",
98          debridge_settings
99      );
100
101     genesis.add_program(
102         "REDACTED",
103         mpl_token_metadata::ID,
104     );
105     println!("debridge settings program id: {:?}",
 ↳ mpl_token_metadata::ID);
106
107     /* #endregion */
108     /* #region CREATE ACCOUNTS */
109     let protocol_authority = Keypair::from_bytes(&[REDACTED]).
 ↳ unwrap();
110     let stop_tap = Keypair::from_bytes(&[REDACTED]).unwrap();
111     let fee_beneficiary = Keypair::from_bytes(&[REDACTED]).unwrap
 ↳ ();
112
113     let token_mint = Pubkey::from_str("
 ↳ FVRMrrf9dWu9NfybuQvZPt8JsLihgiWsCDKqRFhqVGq5").unwrap();
114     let (bridge_data, bump_bridge_data) = Pubkey::
 ↳ find_program_address(
115         &[b"BRIDGE", token_mint.as_ref()],
116         &debridge_settings,
117     );
118     let (chain_support_info, bump_bridge_data) = Pubkey::
 ↳ find_program_address(
119         &[b"CHAIN_SUPPORT_INFO", &ETH_CHAIN_ID.to_bytes()],
120         &debridge_settings,
121     );
122     let (mint_authority, bump_mint_authority) = Pubkey::
 ↳ find_program_address(
123         &[b"BRIDGE_MINTER", bridge_data.as_ref()],
124         &debridge_program,
125     );
126
```

```
127     let (token_metadata, bump_token_metadata) = Pubkey::
 ↳ find_program_address(
128         &[
129             b"metadata",
130             mpl_token_metadata::ID.as_ref(),
131             &token_mint.as_ref(),
132         ],
133         &mpl_token_metadata::ID,
134     );
135 
136     let token_mint_malicious = Pubkey::from_str("8
 ↳ NVwv2oRDtBNBeqrdWdeLGb6T6eYgBmF8QJuzXPjrJim").unwrap();
137     let (bridge_data_malicious, bump_bridge_data) = Pubkey::
 ↳ find_program_address(
138         &[b"BRIDGE", token_mint_malicious.as_ref()],
139         &debridge_settings_malicious,
140     );
141     let (chain_support_info_malicious, bump_bridge_data) = Pubkey
 ↳ ::find_program_address(
142         &[b"CHAIN_SUPPORT_INFO", &ETH_CHAIN_ID.to_bytes()],
143         &debridge_settings_malicious,
144     );
145     let (mint_authority_malicious, bump_mint_authority) = Pubkey::
 ↳ find_program_address(
146         &[b"BRIDGE_MINTER", bridge_data_malicious.as_ref()],
147         &debridge_program_malicious,
148     );
149 
150     let staking_wallet = Pubkey::from_str("
 ↳ ufXtxtzcJZAHzZCdHfgUkdCHvztfTAEjsGCFYfrQLnm").unwrap();
151     let staking_wallet_malicious = Pubkey::from_str("
 ↳ Dg8oCYoEvzrQr49J3GRhT9CDaJ4uPVBJwwYsFZ8y9EP8").unwrap();
152 
153     genesis.add_account_with_lamports(
154         protocol_authority.pubkey(),
155         system_program::ID,
156         10_000_000_000_000_000,
157     );
158 
159     genesis.add_account_with_lamports(
160         stop_tap.pubkey(),
161         system_program::ID,
162         10_000_000_000_000_000,
163     );
```

```
164
165     genesis.add_account_with_lamports(
166         fee_beneficiary.pubkey(),
167         system_program::ID,
168         10_000_000_000_000_000,
169     );
170
171     genesis.add_token_mint(native_token_mint, None, 1, 0, None);
172     genesis.add_token_mint(token_mint, Some(mint_authority), 10
 ↳ _000_000_000, 0, None); //comment this line if using mint bridge
173     genesis.add_token_mint(token_mint_malicious, Some(
 ↳ mint_authority_malicious), 10_000_000_000, 0, None); //comment
 ↳ this line if using mint bridge
174
175     // ADD TOKEN ACCOUNTS
176     genesis.add_token_account(
177         staking_wallet,
178         token_mint,
179         mint_authority,
180         10_000_000_000,
181         None,
182         None,
183         0,
184         None,
185     );
186
187     genesis.add_token_account(
188         staking_wallet_malicious,
189         token_mint_malicious,
190         mint_authority_malicious,
191         10_000_000_000,
192         None,
193         None,
194         0,
195         None,
196     );
197
198     let sender = Keypair::from_bytes(&[REDACTED])
199     .unwrap();
200     let sender_wallet = Pubkey::from_str("8
 ↳ rcFeCFmy5aK1EroGCmcLyGmgfFXL7vPdZqhobbirGpo").unwrap();
201     let sender_wallet_malicious = Pubkey::from_str("
 ↳ FHnSKd6cZHZN2odtmkGLdQurHvK5XPVrxmP9mt1qgwEC").unwrap();
202     genesis.add_token_account(
```

```
203         sender_wallet,
204         token_mint,
205         sender.pubkey(),
206         10,
207         None,
208         None,
209         0,
210         None,
211     );
212     genesis.add_token_account(
213         sender_wallet_malicious,
214         token_mint_malicious,
215         sender.pubkey(),
216         10,
217         None,
218         None,
219         0,
220         None,
221     );
222     genesis.add_account_with_lamports(sender.pubkey(),
 ↳ system_program::ID, 10_000_000_000_000_000);
223
224     let (test_validator, validator_kp) = genesis.start_async().
 ↳ await;
```

Then transactions were send including real and fake transfers.

Sending fake and real instructions in the same transaction:

Listing 13: solana-contracts/programs/debridge/tests/send-with-fake-2.rs (Line 181)

```
181 let url = "http://localhost:8899".to_string();
182  let mut rpc_client = RpcClient::new(url);
183
184  // real IX
185  let mut real_ix = Instruction::new_with_bytes(
186      debridge_program::ID,
187      debridge_program::instruction::Send {
188          amount: 1,
189          target_chain_id: ETH_CHAIN_ID.to_bytes(),
190          receiver: vec![0x00],
191          is_use_asset_fee: false,
```

```
192             submission_params: None,
193             referral_code: None,
194         }
195         .data()
196         .as_ref(),
197         debridge_program::accounts::Sending {
198             nonce_storage,
199             bridge_ctx: debridge_program::accounts::BridgeCtx {
200                 bridge: bridge_data,
201                 token_mint,
202                 staking_wallet,
203                 mint_authority,
204                 chain_support_info,
205                 settings_program: debridge_settings_program::ID,
206                 spl_token_program: spl_token::id(),
207             },
208             state_ctx: debridge_program::accounts::StateCtx {
209                 state: state_settings,
210                 fee_beneficiary: fee_beneficiary.pubkey(),
211             },
212             send_from_wallet: sender_wallet,
213             send_from: sender.pubkey(),
214             system_program: system_program::ID,
215             external_call_storage: Pubkey::new_unique(),
216             external_call_meta: Pubkey::new_unique(),
217             discount: no_discount,
218             bridge_fee,
219         }
220         .to_account_metas(Some(false)),
221     );
222
223     // fake ix
224     let mut fake_ix = Instruction::new_with_bytes(
225         debridge_settings_malicious,
226         debridge_settings_program::instruction::InitializeSendBridge
     ↳ {}
227             .data()
228             .as_ref(),
229         debridge_settings_program::accounts::InitSendBridge {
230             bridge_data: bridge_data_malicious,
231             bridge_id_map: Pubkey::from_str("
     ↳ BMtGiWCJCmsCukUMT2EJwLofBwKMHT79P2nDU3hQs2Tf").unwrap(),
232             state: state_settings_malicious,
233             staking_wallet: staking_wallet_malicious,
```

```
234             token_mint: token_mint_malicious,
235             mint_authority: mint_authority_malicious,
236             system_program: system_program::ID,
237             token_program: spl_token::id(),
238             payer: protocol_authority.pubkey(),
239         }
240         .to_account_metas(Some(false)),
241     );
242
243     let mut tx =  transaction_from_instructions(&mut rpc_client, &[
    ↳ fake_ix, real_ix], &sender, vec![&sender, &protocol_authority]).
    ↳ await.unwrap();
244
245     rpc_client.send_and_confirm_transaction(&tx).unwrap();
```

Sending fake and real instructions in different transactions:

Listing 14: solana-contracts/programs/debridge/tests/send-with-fake.rs (Line 181)

```
181 let url = "http://localhost:8899".to_string();
182 let mut rpc_client = RpcClient::new(url);
183
184 // real IX
185 let mut real_ix = Instruction::new_with_bytes(
186     debridge_program::ID,
187     debridge_program::instruction::Send {
188         amount: 1,
189         target_chain_id: ETH_CHAIN_ID.to_bytes(),
190         receiver: vec![0x00],
191         is_use_asset_fee: false,
192         submission_params: None,
193         referral_code: None,
194     }
195     .data()
196     .as_ref(),
197     debridge_program::accounts::Sending {
198         nonce_storage,
199         bridge_ctx: debridge_program::accounts::BridgeCtx {
200             bridge: bridge_data,
201             token_mint,
202             staking_wallet,
203             mint_authority,
```

```
            chain_support_info,
            settings_program: debridge_settings_program::ID,
            spl_token_program: spl_token::id(),
        },
        state_ctx: debridge_program::accounts::StateCtx {
            state: state_settings,
            fee_beneficiary: fee_beneficiary.pubkey(),
        },
        send_from_wallet: sender_wallet,
        send_from: sender.pubkey(),
        system_program: system_program::ID,
        external_call_storage: Pubkey::new_unique(),
        external_call_meta: Pubkey::new_unique(),
        discount: no_discount,
        bridge_fee,
    }
    .to_account_metas(Some(false)),
);

// fake ix
let mut fake_ix = Instruction::new_with_bytes(
    debridge_program_malicious,
    debridge_program::instruction::Send {
        amount: 1,
        target_chain_id: ETH_CHAIN_ID.to_bytes(),
        receiver: vec![0x00],
        is_use_asset_fee: false,
        submission_params: None,
        referral_code: None,
    }
    .data()
    .as_ref(),
    debridge_program::accounts::Sending {
        nonce_storage: nonce_storage_malicious,
        bridge_ctx: debridge_program::accounts::BridgeCtx {
            bridge: bridge_data_malicious,
            token_mint: token_mint_malicious,
            staking_wallet: staking_wallet_malicious,
            mint_authority: mint_authority_malicious,
            chain_support_info: chain_support_info_malicious,
            settings_program: debridge_settings_malicious,
            spl_token_program: spl_token::id(),
        },
        state_ctx: debridge_program::accounts::StateCtx {
```

```
            state: state_settings_malicious,
            fee_beneficiary: fee_beneficiary.pubkey(),
        },
        send_from_wallet: sender_wallet_malicious,
        send_from: sender.pubkey(),
        system_program: system_program::ID,
        external_call_storage: Pubkey::new_unique(),
        external_call_meta: Pubkey::new_unique(),
        discount: no_discount_malicious,
        bridge_fee: bridge_fee_malicious,
    }
    .to_account_metas(Some(false)),
);

let mut tx =  transaction_from_instructions(&mut rpc_client, &[real_ix], &sender, vec![&sender])
    .await
    .unwrap();

rpc_client.send_and_confirm_transaction(&tx).unwrap();

let mut tx =  transaction_from_instructions(&mut rpc_client, &[fake_ix], &sender, vec![&sender])
.await
.unwrap();

rpc_client.send_and_confirm_transaction(&tx).unwrap();
```

Only real transfer was parsed and stored:

```
}
{"timestamp":"2023-04-20T17:36:12.156648Z","level":"INFO","message":"Program \"TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA\" at level 2, consumed 4645, all: 164021","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T17:36:12.157161Z","level":"INFO","message":"Program \"FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc\" at level 2, consumed 12443, all: 154379","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T17:36:12.157554Z","level":"INFO","message":"Program \"FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc\" at level 2, consumed 12443, all: 138557","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T17:36:12.158122Z","level":"INFO","message":"Program \"6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ\" at level 1, consumed 87727, all: 200000","target":"solana_events_parser::log_parser"}
{"timestamp":"2023-04-20T17:36:12.172321Z","level":"WARN","message":"Error while getting external call data: Error { request: None, kind: RpcError(ForUser(\"AccountNotFound: pubkey=1111111QLbz7JHiBTspS962RLKV8GndWFwiEaqKM\")) }","target":"solana_events::consumer"}
{"timestamp":"2023-04-20T17:36:12.172402Z","level":"INFO","message":"Received amount in transaction 54YU9CeRZfS9g2x4egSsrQ1LF2csnWaa28hQNdzRDYaHywm4ZJqHujT1unPy9WR5r13ueEBXWgJNxDzEgDrtXmuG is 0x0000000000000000000000000000000000000000000000000000000000000001","log.target":"debridge_protobuf_rs::debridge_solana_service","log.module_path":"debridge_protobuf_rs::debridge_solana_service","log.file":"/Users/mb/.cargo/git/checkouts/debridge-protobuf-rs-06d3b571b792a870/1f5ca1b/src/lib.rs","log.line":149,"target":"debridge_protobuf_rs::debridge_solana_service"}
{"timestamp":"2023-04-20T17:36:12.172671Z","level":"INFO","message":"Processing send event: 0x63109602a1f0206d86eeae837e7314d42dcccbb1622ea68869b07bf2e37f458f","target":"solana_events::events_storage"}
{"timestamp":"2023-04-20T17:36:12.174722Z","level":"INFO","message":"INSERT INTO send_events (slot_number, …; rows affected: 1, rows returned: 0, elapsed: 520.541µs\n\nINSERT INTO\n  send_events (slot_number, nonce, protobuf_buffer)\nVALUES\n  ($1, $2, $3) ON CONFLICT (nonce) DO\nUPDATE\nSET\n  \"protobuf_buffer\" = $3\nWHERE\n  send_events.nonce = $2\n","log.target":"sqlx::query","log.module_path":"sqlx::query","target":"sqlx::query"}
```

[10/220]

```
Transaction executed in slot 14390:
  Signature: 54YU9CeRZfS9g2x4egSsrQ1LF2csnWaa28hQNdzRDYaHywm4ZJqHujT1unPy9WR5r13ueEBXWgJNxDzEgDrtXmuG
  Status: Ok
  Log Messages:
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ invoke [1]
    Program log: Instruction: Send
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: Transfer
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 164021 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]
    Program log: Instruction: StakeNativeFee
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 154379 compute units
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc invoke [2]
    Program log: Instruction: StakeBalance
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc consumed 12443 of 138557 compute units
    Program FrAv5D8pH92Mx9kkU6mqMDHEBDqYCeuhtNhSyUngTavc success
    Program data: 53J9Oon4lMkGAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALAEAAAAAAAAA5AtUAgAAAA==
    Program data: FYTvQJLvpkQAYxCWAqHwIG2G7q6DfnMU1C3My7FiLqaIabB78uN/RY9oVr2q96Bqe3TAwGKSQZsAVfuehEN+22BrGmy5L8SwHQEAAAAAAAAAAQAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAGQAAAAAAAAAAAAAAAAAAADufQJ6uDZwC52Alrtp/8OoNLw1/hkFlQUzUPjXXo22RQAAAAA=
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ consumed 87727 of 200000 compute units
    Program 6cQXbaZVmsU1FLT1XhBhi4xx9aRu6Bu2tYKnWmZRUChJ success
Transaction executed in slot 14423:
  Signature: 5kioQnSbnWhG9mjYo73kt869FTmA5JbaqkadWw1x9WJPYKnAFBqGyw5KNf4QkzDU9TLZ9DatiAi45AW9ga6jcWEA
  Status: Ok
  Log Messages:
    Program CjVLsQyCFNpgfMLQ9F5iobzxScWL2FYx3r4sGiHdEJub invoke [1]
    Program log: Instruction: Send
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: Transfer
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4645 of 164021 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program yoUBkA9CyCUQHQNT55KGwbnDEVPRXzQAkfdn5r3xRvj invoke [2]
    Program log: Instruction: StakeNativeFee
```

Real

Fake

Parsing multiple transactions in a short time

In this scenario, multiple valid transfer instructions were sent to do DeBridge program in a very short time:

Listing 15: programs/debridge/tests/send.rs (Line 182)

```rust
let url = "http://localhost:8899".to_string();
    let mut rpc_client = RpcClient::new(url);

    let mut amount = 0;
    for n in 0..5000 {
        println!("SENDING: {}", n);
        amount = amount + 1;
        let mut ix = Instruction::new_with_bytes(
            debridge_program::ID,
            debridge_program::instruction::Send {
                amount,
                target_chain_id: ETH_CHAIN_ID.to_bytes(),
                receiver: vec![0x00],
                is_use_asset_fee: false,
                submission_params: None,
                referral_code: None,
            }
            .data()
            .as_ref(),
            debridge_program::accounts::Sending {
```

```
202                 nonce_storage,
203                 bridge_ctx: debridge_program::accounts::BridgeCtx {
204                     bridge: bridge_data,
205                     token_mint,
206                     staking_wallet,
207                     mint_authority,
208                     chain_support_info,
209                     settings_program: debridge_settings_program::ID
 ↳ ,
210                     spl_token_program: spl_token::id(),
211                 },
212                 state_ctx: debridge_program::accounts::StateCtx {
213                     state: state_settings,
214                     fee_beneficiary: fee_beneficiary.pubkey(),
215                 },
216                 send_from_wallet: sender_wallet,
217                 send_from: sender.pubkey(),
218                 system_program: system_program::ID,
219                 external_call_storage: Pubkey::new_unique(),
220                 external_call_meta: Pubkey::new_unique(),
221                 discount: no_discount,
222                 bridge_fee,
223             }
224             .to_account_metas(Some(false)),
225         );
226 
227         let mut tx = transaction_from_instructions(&mut rpc_client,
 ↳ &[ix], &sender, vec![&sender])
228             .await
229             .unwrap();
230 
231         rpc_client.send_transaction(&tx).unwrap();
232     }
```

Number of events stored before test:

```
postgres=# select * from send_events;
 nonce | slot_number | protobuf_buffer
-------+-------------+-----------------
(0 rows)
```

All send transactions were stored:

```
postgres=# select count(*) from send_events;
 count
-------
  5000
(1 row)
```

Note that this scenario is not an equivalent of stress testing, as tests were performed on the local cluster and own deployment of Events Reader. On production deployment, the service should be deployed with proper settings regarding server performance and scaling mechanisms.

4.2 GRPC Service

Description:

GRPC Service connects to the database, validates and presents events in public API. The design and implementation of the connections between the sample client and service were validated:

- Authorization mechanisms,
- Tests against potential DOS/DDOS (Denial of Service / Distributed Denial of Service) attacks,
- Possibilities of malicious code injection to GRPC Service,
- Possibilities of information disclosure vulnerabilities.

Results:

Observations and proof of concept were presented in the HAL-02 issue in the previous section of the report.

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 called 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-2023-0023	openssl	openssl SubjectAlternativeName and ExtendedKeyUsage::other allow arbitrary file read
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2023-0018	remove dir all	Race Condition Enabling Link Following and Time-of-check Time-of-use (TOCTOU)

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

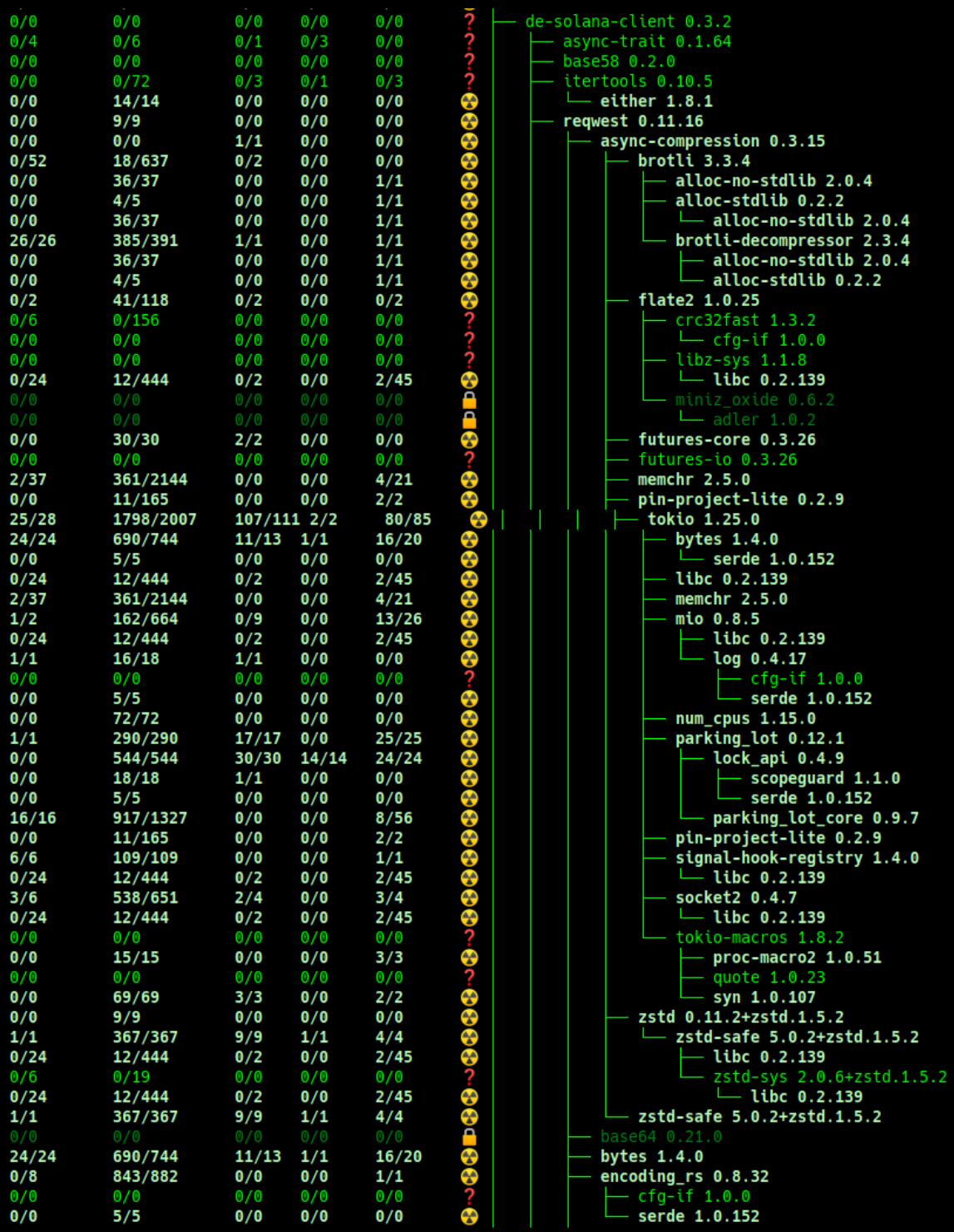
AUTOMATED TESTING

Results:

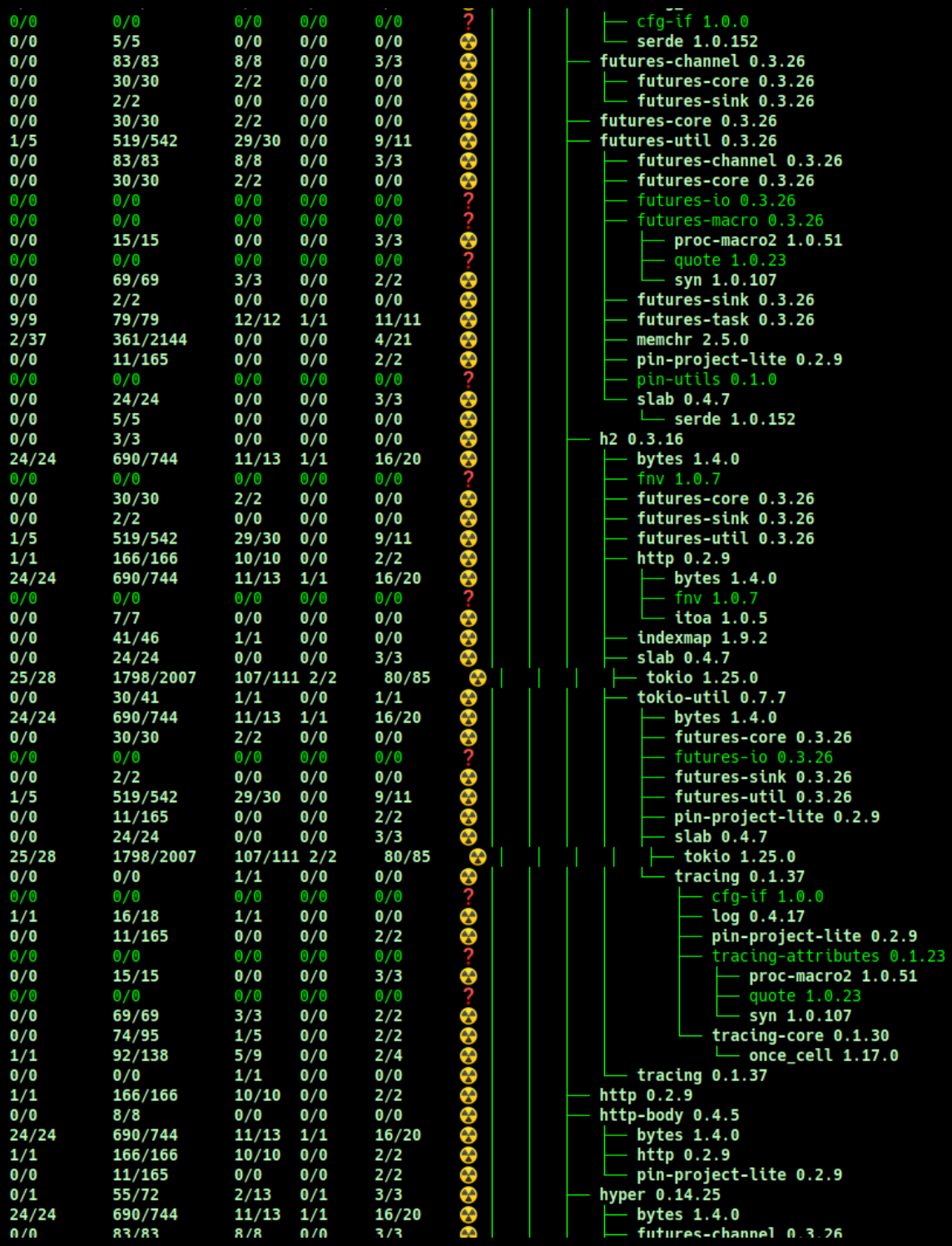
```
Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ solana-events 0.0.2
15/18      453/460      3/3    0/0     12/12    ☢  ├── anyhow 1.0.69
0/4        0/6          0/1    0/3     0/0      ❓ ├── async-trait 0.1.64
0/0        15/15        0/0    0/0     3/3      ☢  │   ├── proc-macro2 1.0.51
0/0        4/4          0/0    0/0     0/0      ☢  │   │   └── unicode-ident 1.0.6
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── quote 1.0.23
0/0        15/15        0/0    0/0     3/3      ☢  │   │   └── proc-macro2 1.0.51
0/0        69/69        3/3    0/0     2/2      ☢  │   └── syn 1.0.107
0/0        15/15        0/0    0/0     3/3      ☢  │       ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │       ├── quote 1.0.23
0/0        4/4          0/0    0/0     0/0      ☢  │       └── unicode-ident 1.0.6
0/0        0/0          0/0    0/0     0/0      🔒 ├── base64 0.13.1
0/0        2/48         2/2    0/0     0/0      ☢  ├── chrono 0.4.23
0/2        1/147        0/0    0/0     0/1      ☢  │   ├── iana-time-zone 0.1.53
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── num-integer 0.1.45
0/0        6/12         0/0    0/0     0/0      ☢  │   │   └── num-traits 0.2.15
0/0        6/12         0/0    0/0     0/0      ☢  │   ├── num-traits 0.2.15
0/0        5/5          0/0    0/0     0/0      ☢  │   ├── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ❓ │   │   └── serde_derive 1.0.152
0/0        15/15        0/0    0/0     3/3      ☢  │   │       ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │   │       ├── quote 1.0.23
0/0        69/69        3/3    0/0     2/2      ☢  │   │       └── syn 1.0.107
1/1        221/221      0/0    0/0     0/0      ☢  │   └── time 0.1.45
0/24       12/444       0/2    0/0     2/45     ☢  │       └── libc 0.2.139
0/0        0/0          0/0    0/0     0/0      ❓ ├── confique 0.2.3-alpha.4
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── confique-macro 0.0.7
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── heck 0.3.3
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   └── unicode-segmentation 1.10.1
0/0        15/15        0/0    0/0     3/3      ☢  │   │   ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── quote 1.0.23
0/0        69/69        3/3    0/0     2/2      ☢  │   │   └── syn 1.0.107
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── json5 0.4.1
2/2        71/71        0/0    0/0     2/2      ☢  │   │   ├── pest 2.5.5
2/37       361/2144     0/0    0/0     4/21     ☢  │   │   │   ├── memchr 2.5.0
0/24       12/444       0/2    0/0     2/45     ☢  │   │   │   │   └── libc 0.2.139
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   ├── serde 1.0.152
0/0        7/7          0/0    0/0     0/0      ☢  │   │   │   ├── serde_json 1.0.92
0/0        41/46        1/1    0/0     0/0      ☢  │   │   │   │   ├── indexmap 1.9.2
1/1        1223/1367    21/24  1/1     62/69    ☢  │   │   │   │   │   ├── hashbrown 0.12.3
0/0        26/30        0/0    0/0     0/0      ☢  │   │   │   │   │   │   ├── ahash 0.7.6
2/4        52/175       1/1    0/0     3/3      ☢  │   │   │   │   │   │   │   ├── getrandom 0.2.8
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │   │   │   ├── cfg-if 1.0.0
0/24       12/444       0/2    0/0     2/45     ☢  │   │   │   │   │   │   │   │   └── libc 0.2.139
1/1        92/138       5/9    0/0     2/4      ☢  │   │   │   │   │   │   │   └── once_cell 1.17.0
0/2        0/36         0/2    0/1     0/3      ❓ │   │   │   │   │   │   │       ├── critical-section 1.1.1
16/16      917/1327     0/0    0/0     8/56     ☢  │   │   │   │   │   │   │       └── parking_lot_core 0.9.7
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │   │           ├── cfg-if 1.0.0
0/24       12/444       0/2    0/0     2/45     ☢  │   │   │   │   │   │   │           ├── libc 0.2.139
2/2        64/75        5/5    1/1     1/1      ☢  │   │   │   │   │   │   │           └── petgraph 0.6.3
0/0        70/70        0/0    0/0     0/0      ☢  │   │   │   │   │   │   │               ├── fixedbitset 0.4.2
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   │   │               │   └── serde 1.0.152
0/0        41/46        1/1    0/0     0/0      ☢  │   │   │   │   │   │   │               ├── indexmap 1.9.2
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   │   │               ├── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │   │               └── serde_derive 1.0.152
1/1        399/399      7/7    1/1     13/13    ☢  │   │   │   │   │   │   │           └── smallvec 1.10.0
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   │   │               └── serde 1.0.152
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   │   └── serde 1.0.152
4/6        437/1158     4/10   1/1     13/26    ☢  │   │   │   │   │   ├── bumpalo 3.12.0
6/6        659/659      5/5    0/0     3/3      ☢  │   │   │   │   │   ├── rayon 1.6.1
0/0        14/14        0/0    0/0     0/0      ☢  │   │   │   │   │   │   ├── either 1.8.1
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   │   │   └── serde 1.0.152
4/6        437/1158     4/10   1/1     13/26    ☢  │   │   │   │   │   ├── bumpalo 3.12.0
6/6        659/659      5/5    0/0     3/3      ☢  │   │   │   │   │   ├── rayon 1.6.1
0/0        14/14        0/0    0/0     0/0      ☢  │   │   │   │   │   │   ├── either 1.8.1
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   │   │   └── serde 1.0.152
7/7        657/660      5/5    0/0     33/33    ☢  │   │   │   │   │   │   └── rayon-core 1.10.2
2/2        485/494      6/7    0/0     12/14    ☢  │   │   │   │   │   │       ├── crossbeam-channel 0.5.6
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │       │   ├── cfg-if 1.0.0
4/4        94/94        16/16  0/0     3/3      ☢  │   │   │   │   │   │       │   └── crossbeam-utils 0.8.14
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │       │       └── cfg-if 1.0.0
0/0        453/453      6/6    0/0     6/6      ☢  │   │   │   │   │   │       ├── crossbeam-deque 0.8.2
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │       │   ├── cfg-if 1.0.0
3/3        448/460      11/11  0/0     29/29    ☢  │   │   │   │   │   │       │   ├── crossbeam-epoch 0.9.13
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │       │   │   ├── cfg-if 1.0.0
4/4        94/94        16/16  0/0     3/3      ☢  │   │   │   │   │   │       │   │   ├── crossbeam-utils 0.8.14
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │       │   │   ├── memoffset 0.7.1
0/0        18/18        1/1    0/0     0/0      ☢  │   │   │   │   │   │       │   │   └── scopeguard 1.1.0
4/4        94/94        16/16  0/0     3/3      ☢  │   │   │   │   │   │       │   └── crossbeam-utils 0.8.14
4/4        94/94        16/16  0/0     3/3      ☢  │   │   │   │   │   │       ├── crossbeam-utils 0.8.14
0/0        72/72        0/0    0/0     0/0      ☢  │   │   │   │   │   │       └── num_cpus 1.15.0
0/24       12/444       0/2    0/0     2/45     ☢  │   │   │   │   │   │           └── libc 0.2.139
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   │   └── serde 1.0.152
6/6        659/659      5/5    0/0     3/3      ☢  │   │   │   │   ├── rayon 1.6.1
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   └── serde 1.0.152
0/0        7/7          0/0    0/0     0/0      ☢  │   │   │   ├── itoa 1.0.5
7/9        579/715      0/0    0/0     2/2      ☢  │   │   │   ├── ryu 1.0.12
```

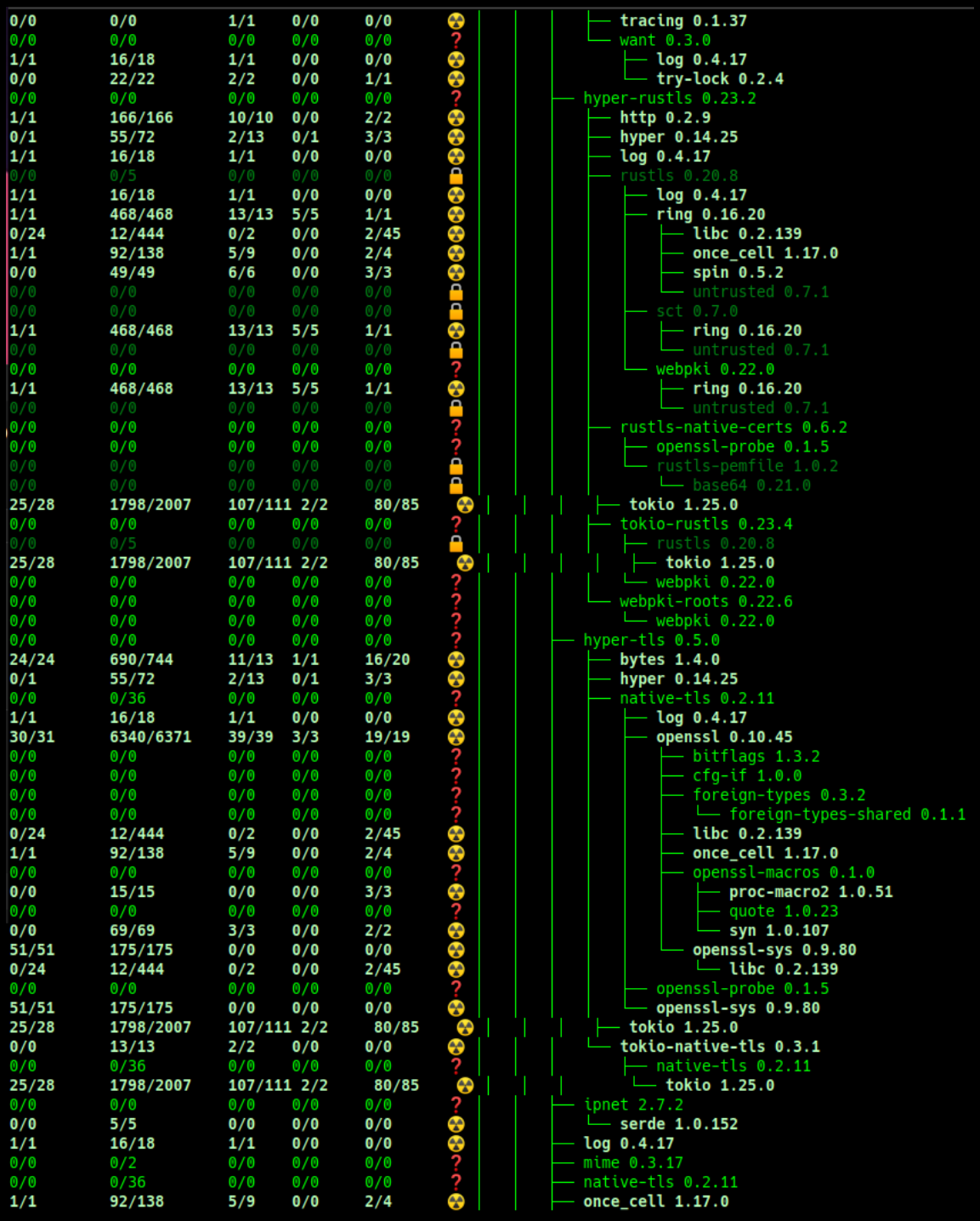
AUTOMATED TESTING



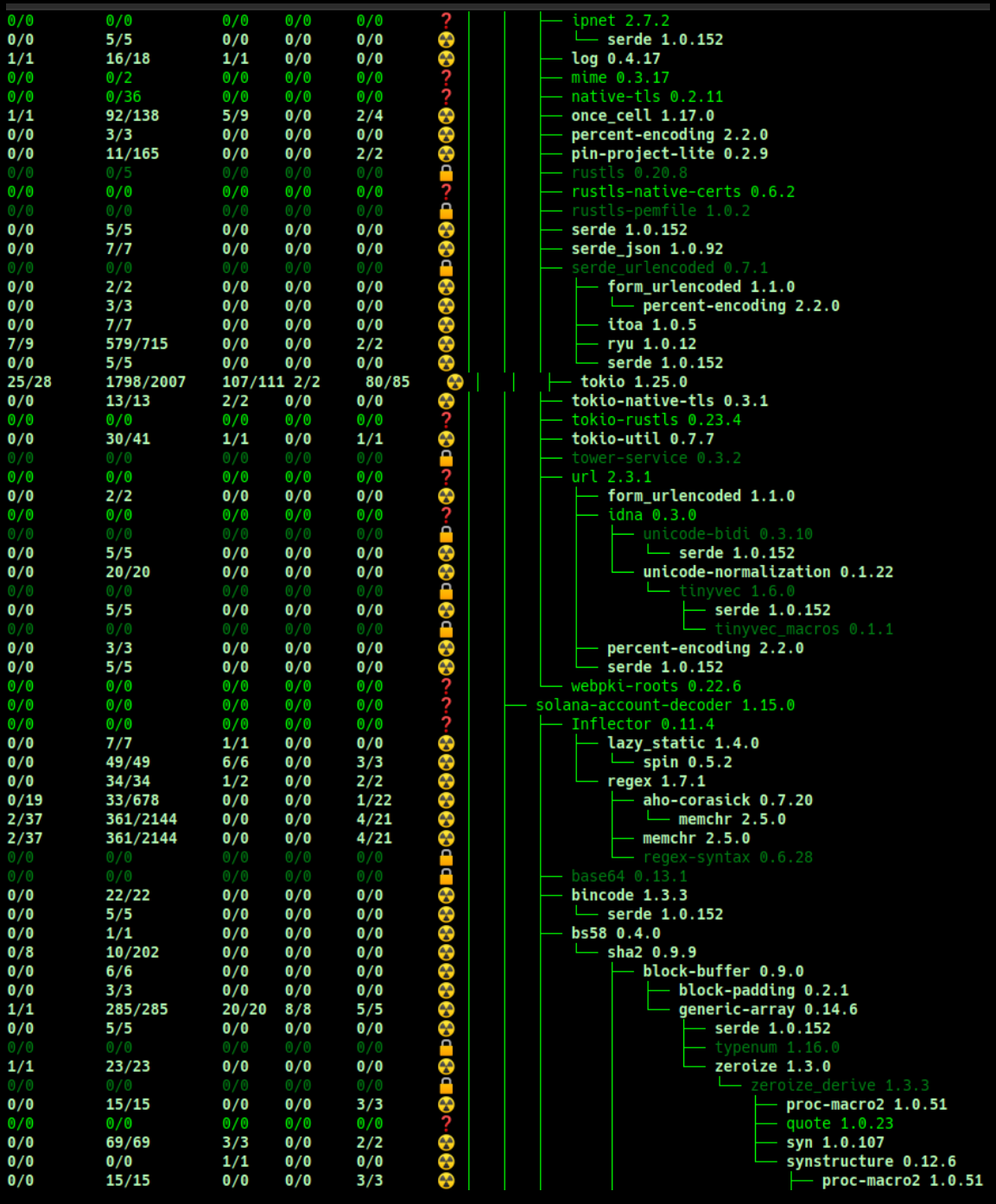
AUTOMATED TESTING



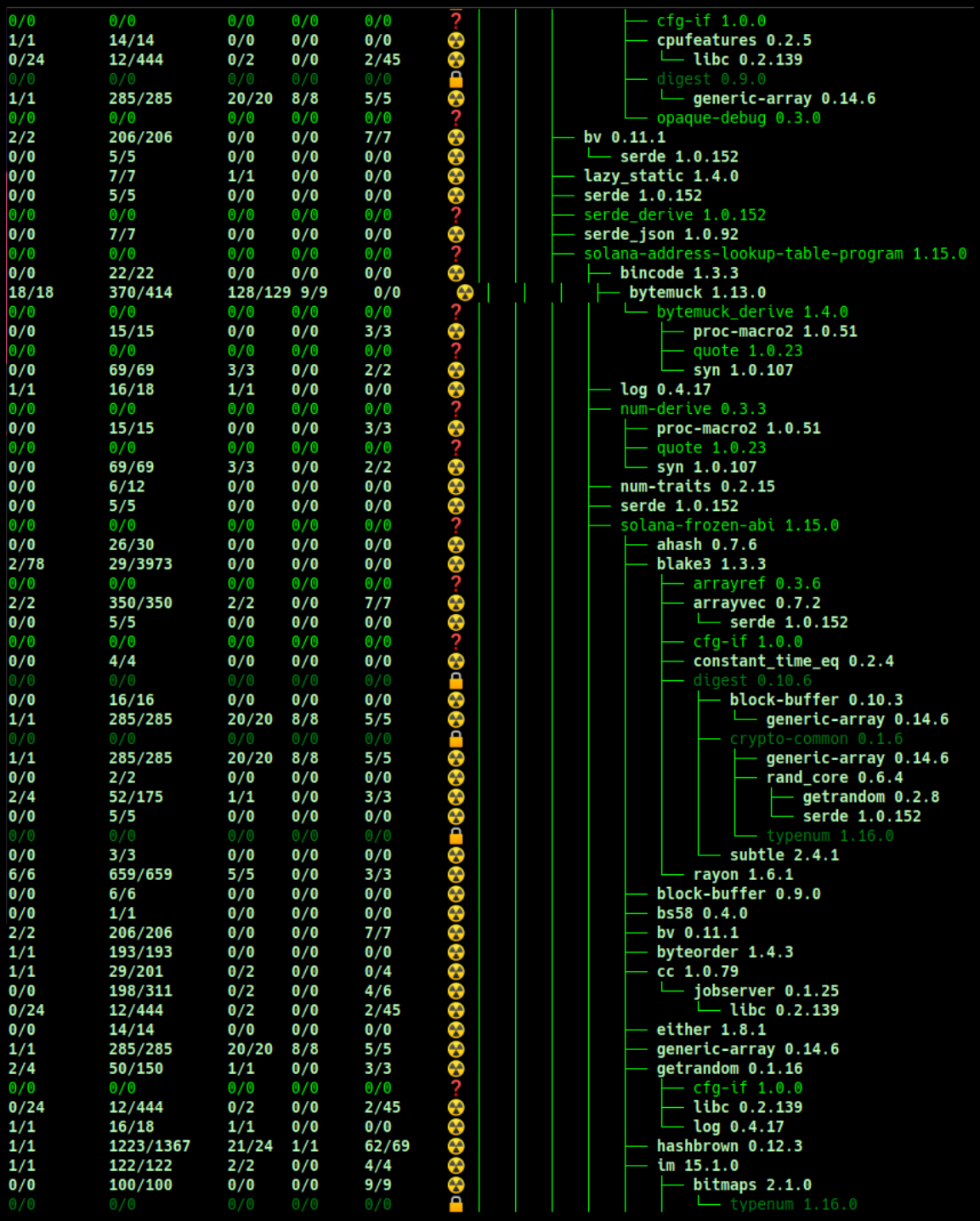
AUTOMATED TESTING



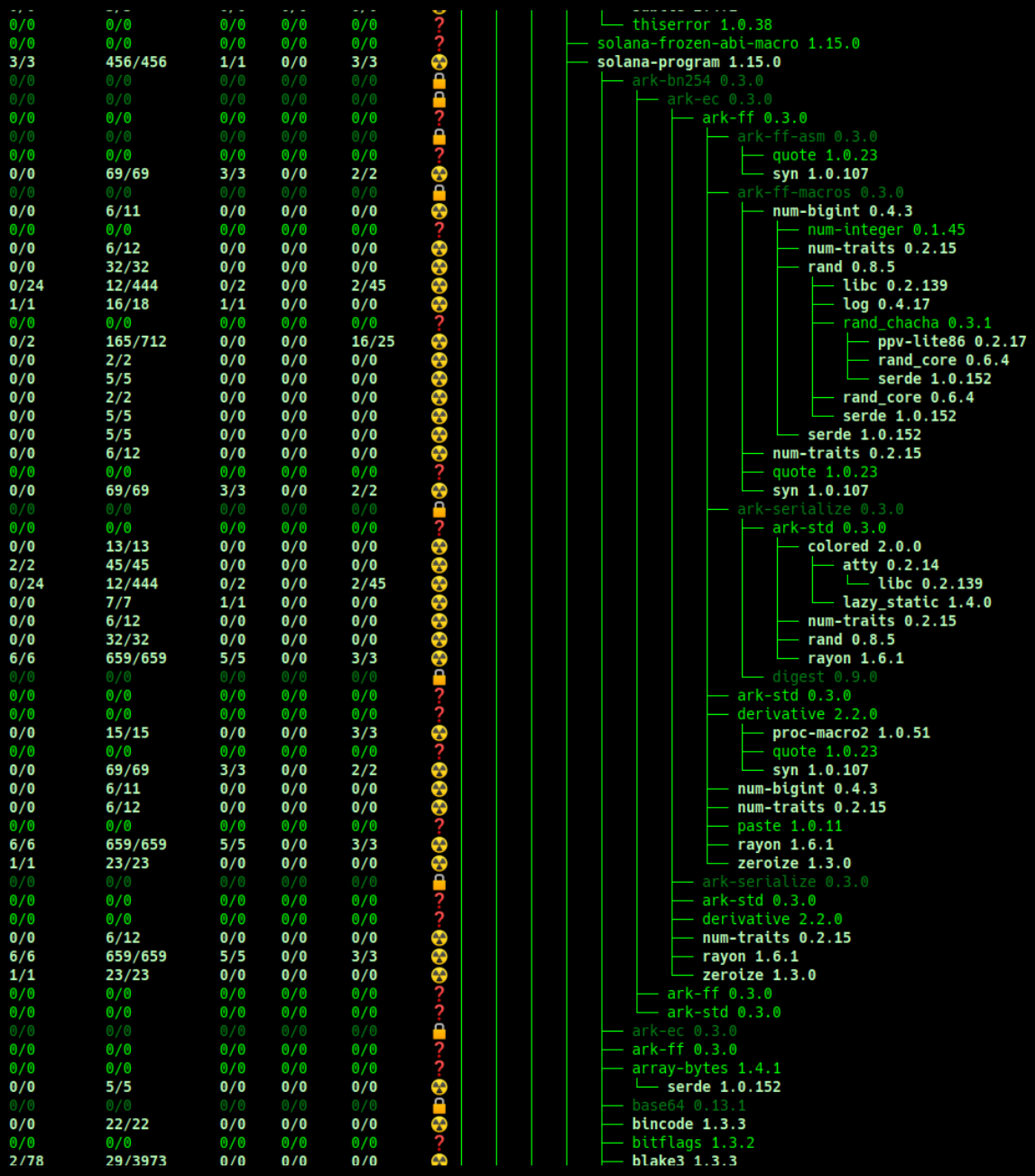
AUTOMATED TESTING



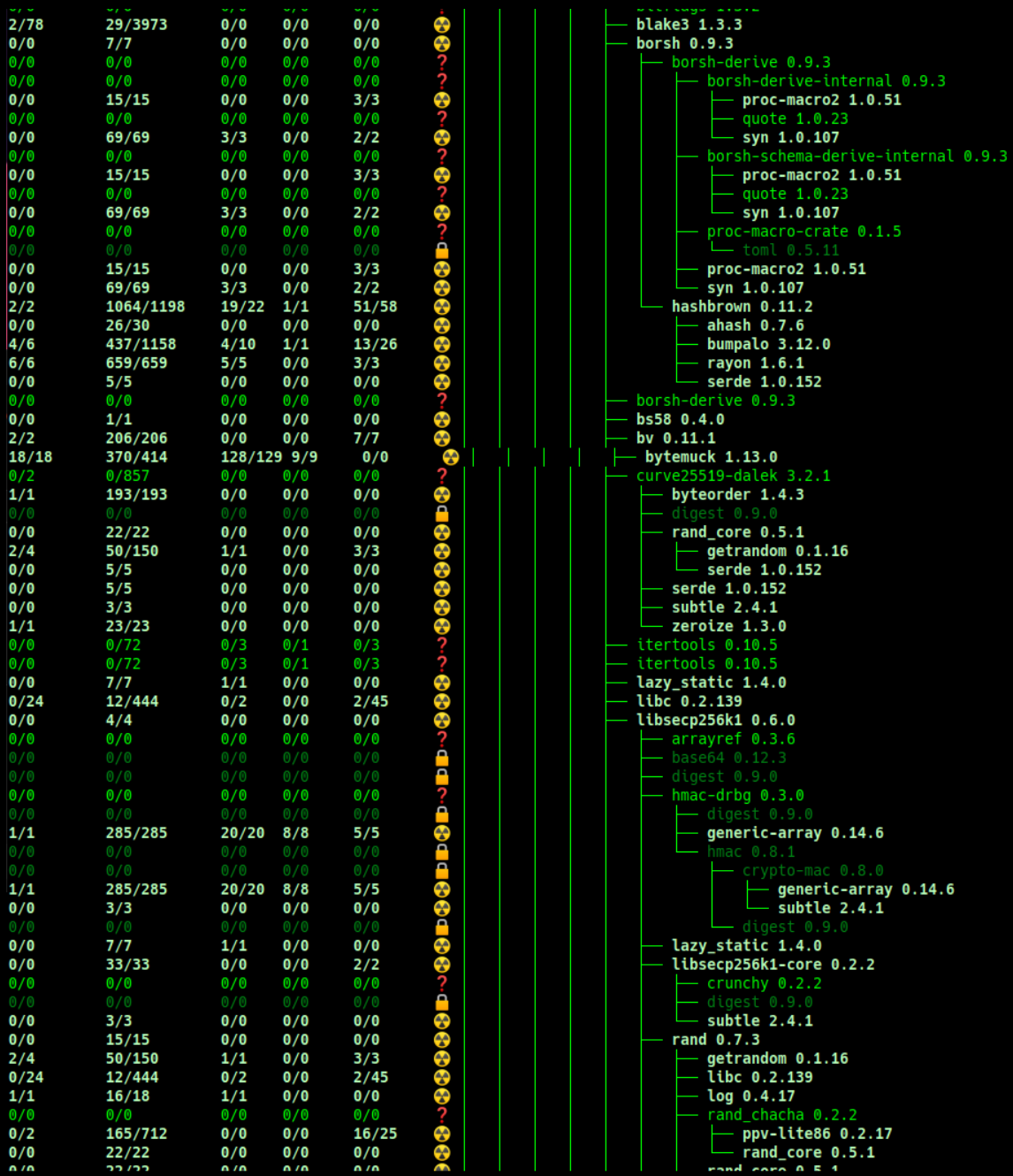
AUTOMATED TESTING



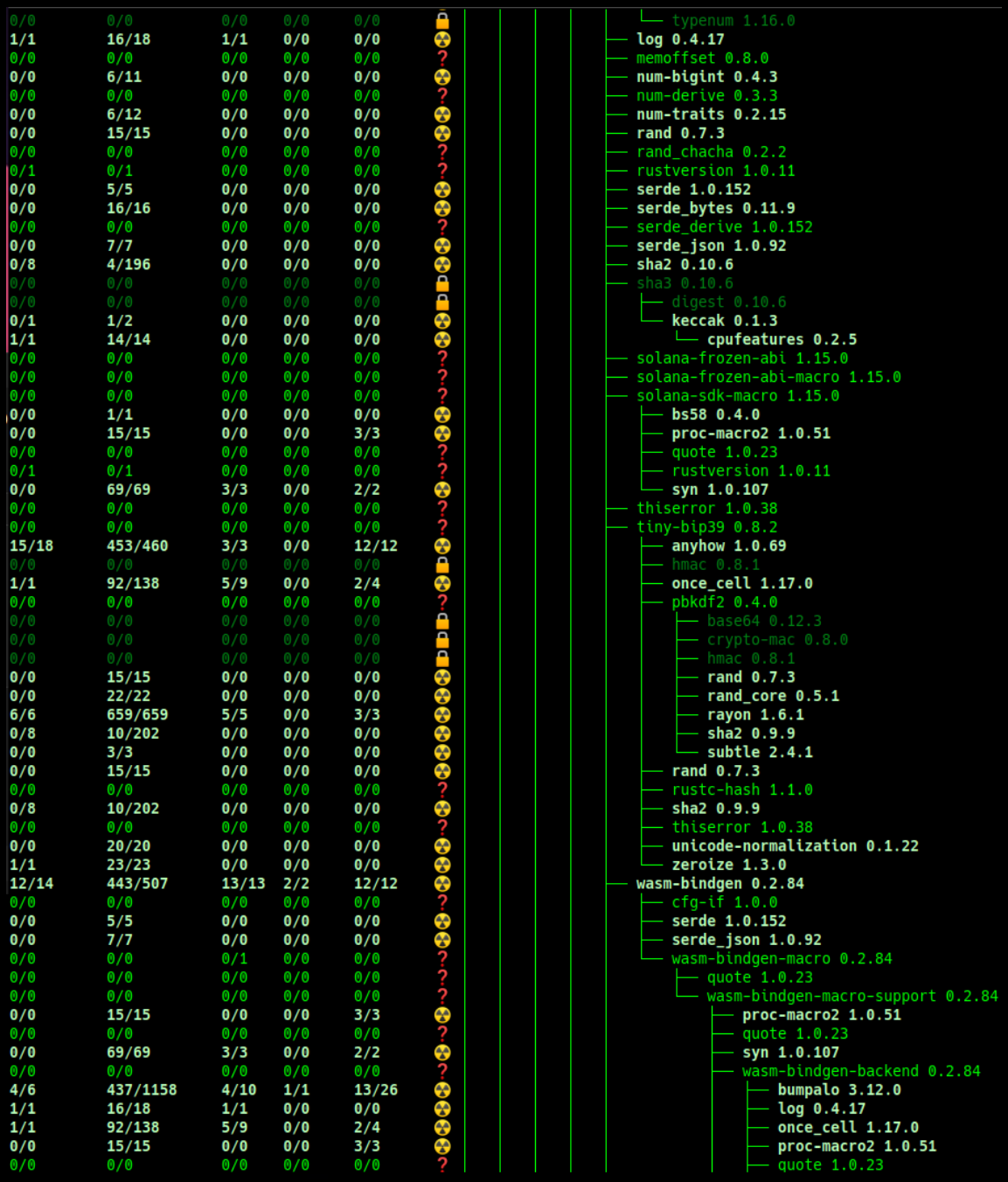
AUTOMATED TESTING



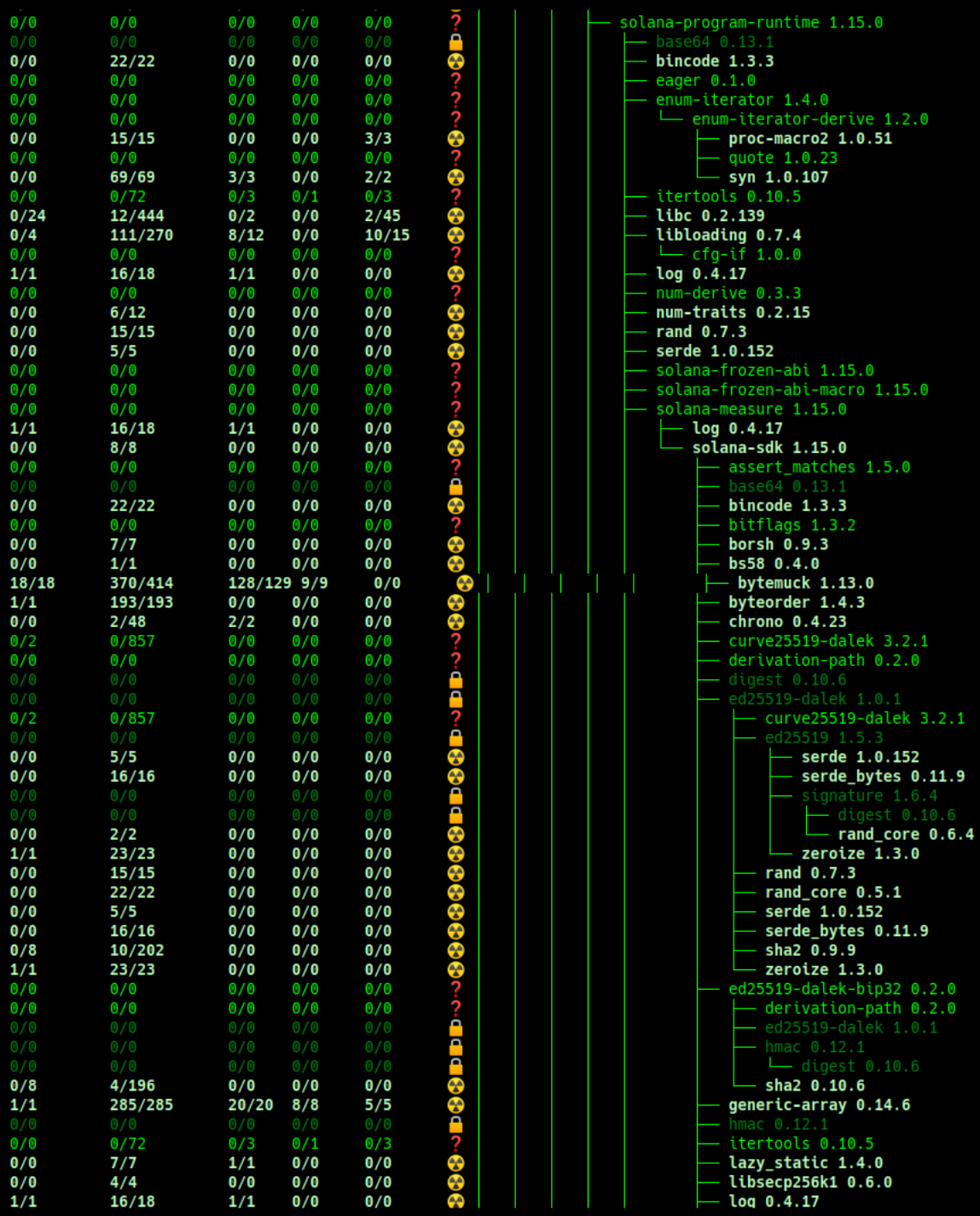
AUTOMATED TESTING

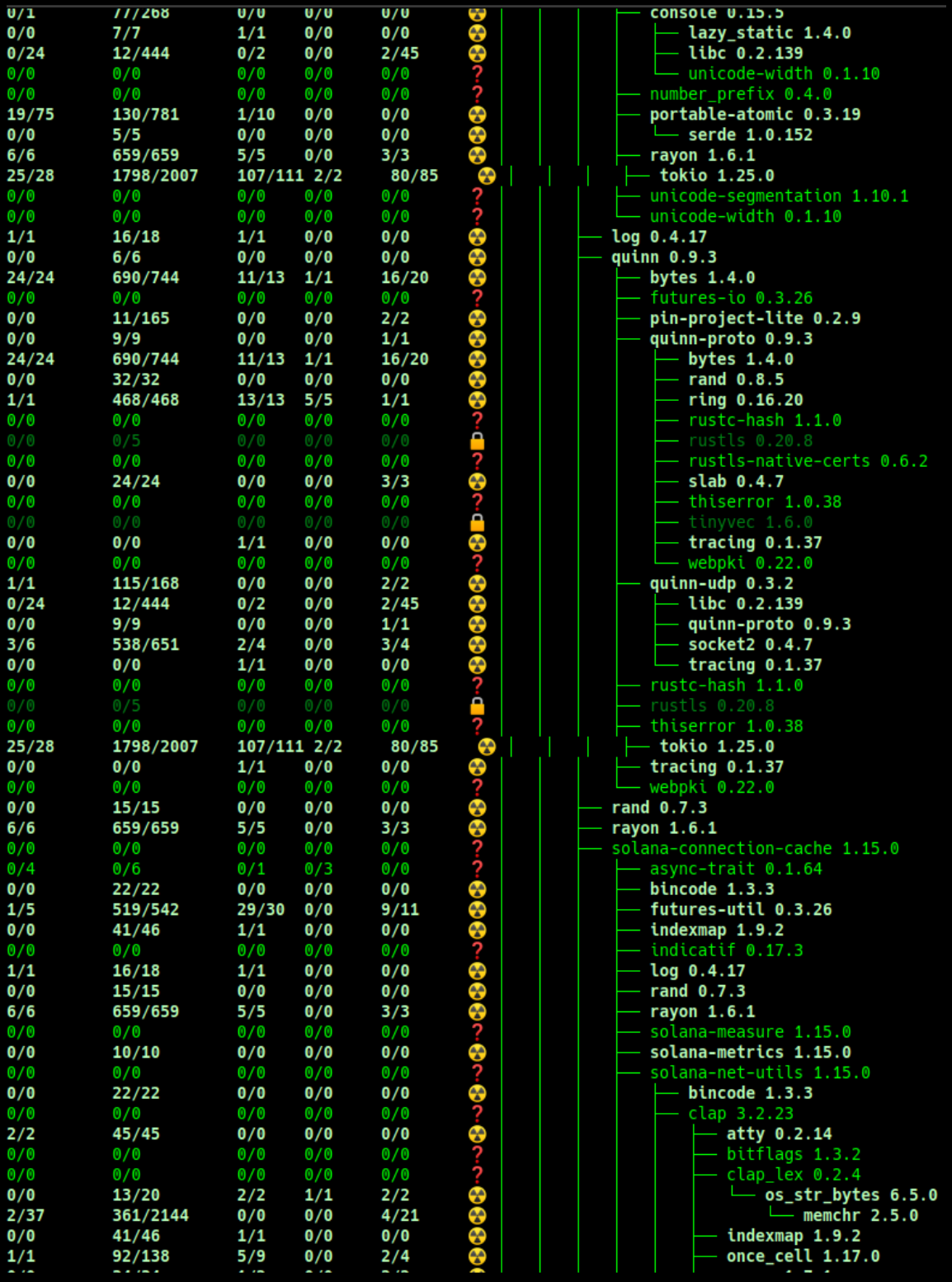


AUTOMATED TESTING

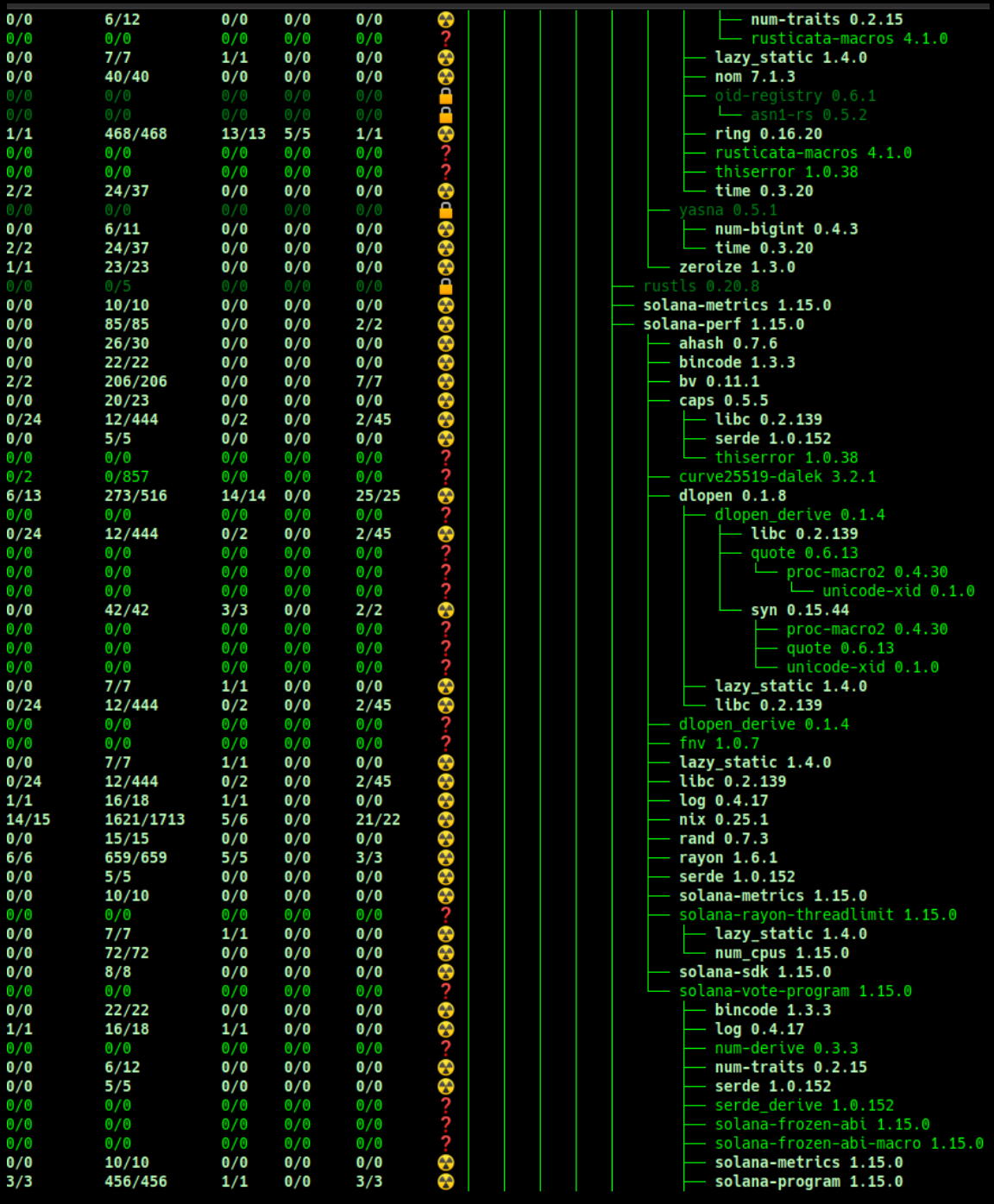


AUTOMATED TESTING

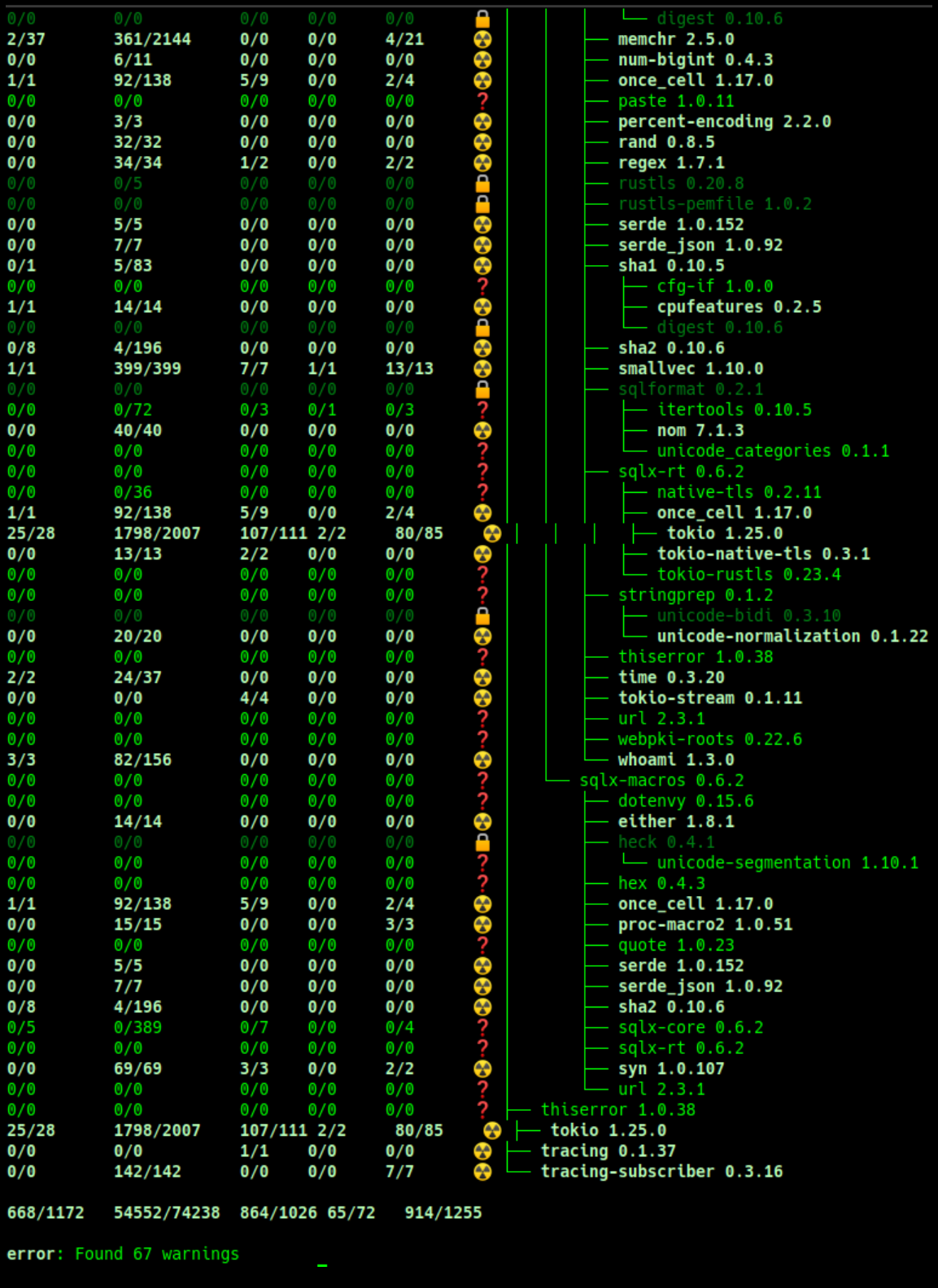



0/1 77/268 0/0 0/0 0/0 ☢ ├── console 0.15.5
0/0 7/7 1/1 0/0 0/0 ☢ │ ├── lazy_static 1.4.0
0/24 12/444 0/2 0/0 2/45 ☢ │ ├── libc 0.2.139
0/0 0/0 0/0 0/0 0/0 ❓ │ └── unicode-width 0.1.10
0/0 0/0 0/0 0/0 0/0 ❓ ├── number_prefix 0.4.0
19/75 130/781 1/10 0/0 0/0 ☢ ├── portable-atomic 0.3.19
0/0 5/5 0/0 0/0 0/0 ☢ │ └── serde 1.0.152
6/6 659/659 5/5 0/0 3/3 ☢ ├── rayon 1.6.1
25/28 1798/2007 107/111 2/2 80/85 ☢ │ └── tokio 1.25.0
0/0 0/0 0/0 0/0 0/0 ❓ ├── unicode-segmentation 1.10.1
0/0 0/0 0/0 0/0 0/0 ❓ └── unicode-width 0.1.10
1/1 16/18 1/1 0/0 0/0 ☢ ├── log 0.4.17
0/0 6/6 0/0 0/0 0/0 ☢ ├── quinn 0.9.3
24/24 690/744 11/13 1/1 16/20 ☢ ├── bytes 1.4.0
0/0 0/0 0/0 0/0 0/0 ❓ ├── futures-io 0.3.26
0/0 11/165 0/0 0/0 2/2 ☢ ├── pin-project-lite 0.2.9
0/0 9/9 0/0 0/0 1/1 ☢ ├── quinn-proto 0.9.3
24/24 690/744 11/13 1/1 16/20 ☢ │ ├── bytes 1.4.0
0/0 32/32 0/0 0/0 0/0 ☢ │ ├── rand 0.8.5
1/1 468/468 13/13 5/5 1/1 ☢ │ ├── ring 0.16.20
0/0 0/0 0/0 0/0 0/0 ❓ │ ├── rustc-hash 1.1.0
0/0 0/5 0/0 0/0 0/0 🔒 │ ├── rustls 0.20.8
0/0 0/0 0/0 0/0 0/0 ❓ │ ├── rustls-native-certs 0.6.2
0/0 24/24 0/0 0/0 3/3 ☢ │ ├── slab 0.4.7
0/0 0/0 0/0 0/0 0/0 ❓ │ ├── thiserror 1.0.38
0/0 0/0 0/0 0/0 0/0 🔒 │ ├── tinyvec 1.6.0
0/0 0/0 1/1 0/0 0/0 ☢ │ ├── tracing 0.1.37
0/0 0/0 0/0 0/0 0/0 ❓ │ └── webpki 0.22.0
1/1 115/168 0/0 0/0 2/2 ☢ ├── quinn-udp 0.3.2
0/24 12/444 0/2 0/0 2/45 ☢ │ ├── libc 0.2.139
0/0 9/9 0/0 0/0 1/1 ☢ │ ├── quinn-proto 0.9.3
3/6 538/651 2/4 0/0 3/4 ☢ │ ├── socket2 0.4.7
0/0 0/0 1/1 0/0 0/0 ☢ │ └── tracing 0.1.37
0/0 0/0 0/0 0/0 0/0 ❓ ├── rustc-hash 1.1.0
0/0 0/5 0/0 0/0 0/0 🔒 ├── rustls 0.20.8
0/0 0/0 0/0 0/0 0/0 ❓ ├── thiserror 1.0.38
25/28 1798/2007 107/111 2/2 80/85 ☢ │ └── tokio 1.25.0
0/0 0/0 1/1 0/0 0/0 ☢ ├── tracing 0.1.37
0/0 0/0 0/0 0/0 0/0 ❓ └── webpki 0.22.0
0/0 15/15 0/0 0/0 0/0 ☢ ├── rand 0.7.3
6/6 659/659 5/5 0/0 3/3 ☢ ├── rayon 1.6.1
0/0 0/0 0/0 0/0 0/0 ❓ ├── solana-connection-cache 1.15.0
0/4 0/6 0/1 0/3 0/0 ❓ │ ├── async-trait 0.1.64
0/0 22/22 0/0 0/0 0/0 ☢ │ ├── bincode 1.3.3
1/5 519/542 29/30 0/0 9/11 ☢ │ ├── futures-util 0.3.26
0/0 41/46 1/1 0/0 0/0 ☢ │ ├── indexmap 1.9.2
0/0 0/0 0/0 0/0 0/0 ❓ │ ├── indicatif 0.17.3
1/1 16/18 1/1 0/0 0/0 ☢ │ ├── log 0.4.17
0/0 15/15 0/0 0/0 0/0 ☢ │ ├── rand 0.7.3
6/6 659/659 5/5 0/0 3/3 ☢ │ ├── rayon 1.6.1
0/0 0/0 0/0 0/0 0/0 ❓ │ ├── solana-measure 1.15.0
0/0 10/10 0/0 0/0 0/0 ☢ │ ├── solana-metrics 1.15.0
0/0 0/0 0/0 0/0 0/0 ❓ │ ├── solana-net-utils 1.15.0
0/0 22/22 0/0 0/0 0/0 ☢ │ │ ├── bincode 1.3.3
0/0 0/0 0/0 0/0 0/0 ❓ │ │ ├── clap 3.2.23
2/2 45/45 0/0 0/0 0/0 ☢ │ │ │ ├── atty 0.2.14
0/0 0/0 0/0 0/0 0/0 ❓ │ │ │ ├── bitflags 1.3.2
0/0 0/0 0/0 0/0 0/0 ❓ │ │ │ ├── clap_lex 0.2.4
0/0 13/20 2/2 1/1 2/2 ☢ │ │ │ │ └── os_str_bytes 6.5.0
2/37 361/2144 0/0 0/0 4/21 ☢ │ │ │ │ └── memchr 2.5.0
0/0 41/46 1/1 0/0 0/0 ☢ │ │ │ ├── indexmap 1.9.2
1/1 92/138 5/9 0/0 2/4 ☢ │ │ │ ├── once_cell 1.17.0

AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN