HALBORN

Rarimo - Solana Bridge

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 12th, 2022 - December 23rd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/09/2022	Guillermo Alvarez
0.2	Document Updates	12/22/2022	Guillermo Alvarez
0.3	Final Draft	12/23/2022	Guillermo Alvarez
0.4	Draft Review	12/23/2022	Isabel Burruezo
0.5	Draft Review	12/23/2022	Piotr Cielas
0.6	Draft Review	12/23/2022	Gabi Urrutia
1.0	Remediation Plan	02/27/2023	Guillermo Alvarez
1.1	Remediation Plan Review	02/27/2023	Isabel Burruezo
1.2	Remediation Plan Review	02/27/2023	Piotr Cielas
1.3	Remediation Plan Review	02/27/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Isabel Burruezo	Halborn	Isabel.Burruezo@halborn.com
Guillermo Alvarez	Halborn	Guillermo.Alvarez@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Rarimo is the next-generation messaging protocol offering cross-chain data, identity, and digital asset management. Rarimo enables the free movement of assets and identities across chains, allowing users and dApps a choice that was not afforded before. With Rarimo, the future of multi-chain ecosystems and the network effect between them is unlocked. There are many advantages seen in using Rarimo. The first is that Rarimo provides high-security guarantees at a constant transfer price (due to the use of TSS). The second is the ease of integrating messaging contracts with existing dApps. This saves users and developers of the application on top of the bridge to understand the technical features of the systems between which such a transfer is made. The third advantage is that the architecture of the protocol allows the implementation of streamlined user flows: one action = result. The user can interact with decentralized applications directly through the messaging protocol using the token of their choice and send only one transaction to the parent network. This feature opens up many use cases such as single transaction cross-chain settlement, cross-chain lending, and staking through a single transaction.

Rarimo engaged Halborn to conduct a security audit on their Solana programs beginning on December 12th, 2022 and ending on December 23rd, 2022 . The security assessment was scoped to the programs provided in the solana-rarimo-bridge-program GitLab repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided 2 weeks for the engagement and assigned 1 full-time security engineer to audit the security of the programs in scope. The security engineer is a blockchain and Solana program security expert with advanced penetration testing and Solana program hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Identify potential security issues within the programs

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by Rarimo. The main ones are the following:
- The Rarimo team accepted the risk of using deprecated functions and stated that will monitor release notes for Metaplex and Associate Token Account programs to ensure that no vulnerabilities affect the bridge.
- All informational findings presented, except one, were fixed by the Rarimo team. Regarding the lack of test coverage the team acknowledged the issue and mentioned that they plan to implement tests in the future.

1.3 TEST APPROACH & METHODOLOGY

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

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

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.

- Finding unsafe Rust code usage (cargo-geiger)
- Scanning dependencies for known vulnerabilities (cargo audit).
- Local runtime testing (solana-test-framework)
- Scanning for common Solana vulnerabilities (soteria)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

Code repositories:

1. Solana Rarimo Bridge

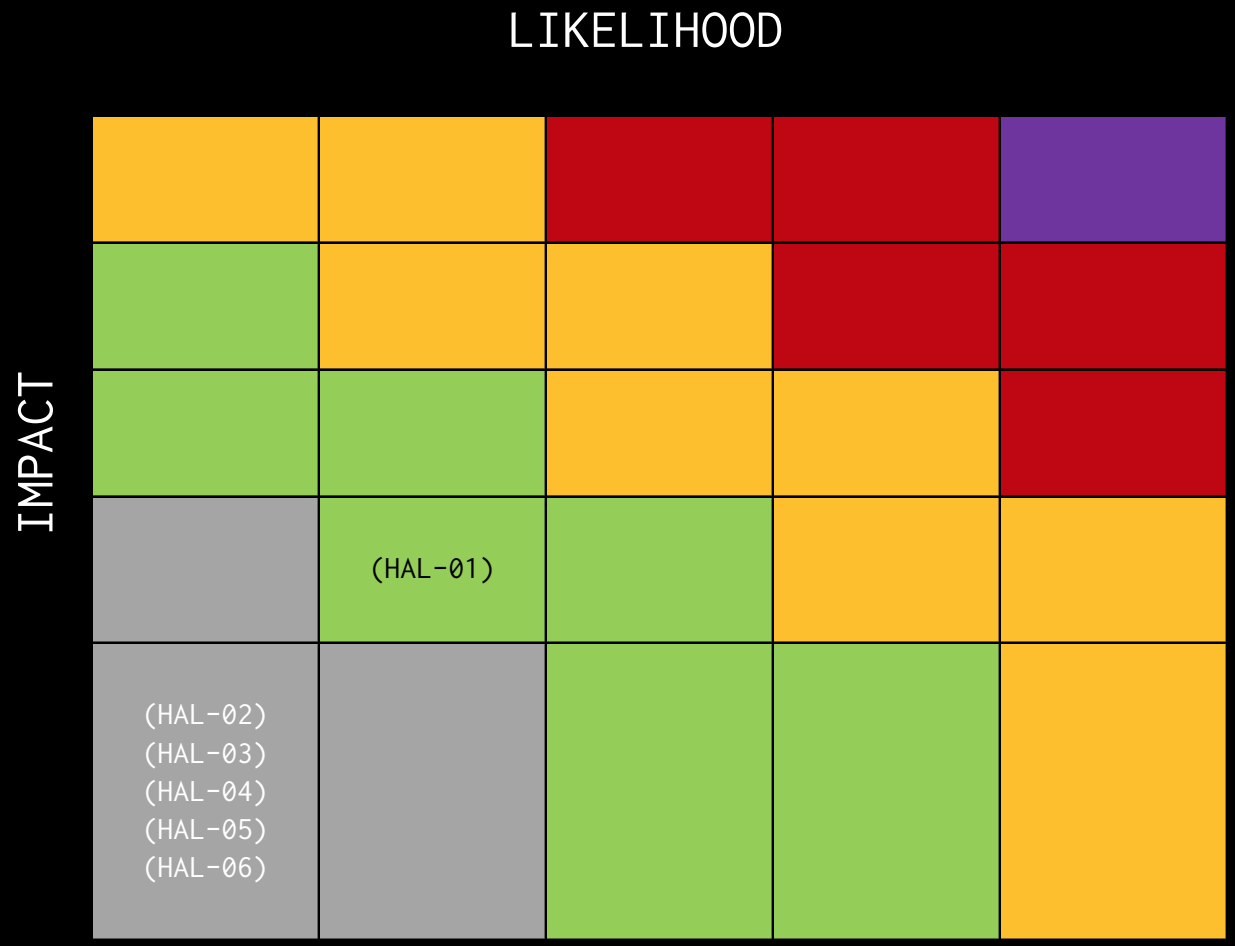
- Repository: solana-rarimo-bridge-program
- Commit ID: 432cda635a7964733c3aa24eb48e554b5f65094f
- Programs in scope:
    1. (solana-rarimo-brigde-program/src/)

Note:
After the initial audit, the repository was migrated to solana-bridge-program, which was used for the remediation plan review.

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	5



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - DEPRECATED FUNCTIONS IN USE	Low	RISK ACCEPTED
HAL-02 - MISSING CARGO OVERFLOW CHECKS	Informational	SOLVED - 02/27/2023
HAL-03 - INTEGER UNDERFLOW	Informational	SOLVED - 02/27/2023
HAL-04 - DEAD CODE	Informational	SOLVED - 02/27/2023
HAL-05 - LACK OF TEST COVERAGE	Informational	ACKNOWLEDGED
HAL-06 - POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	SOLVED - 02/27/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) DEPRECATED FUNCTIONS IN USE - LOW

Description:

Multiple deprecated mpl-token-metadata and spl-associated-token-account functions are used by the program, not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby. Additionally, deprecated functions can be removed in newer releases of the affected dependencies, increasing the risk to the long-term sustainability and integrity of the code.

Code Location:

Listing 1: src/processor.rs (Line 1156)

```
145 fn call_create_metadata<'a>(
146     metadata_account: &AccountInfo<'a>,
147     mint: &AccountInfo<'a>,
148     mint_authority: &AccountInfo<'a>,
149     payer: &AccountInfo<'a>,
150     update_authority: &AccountInfo<'a>,
151     rent: &AccountInfo<'a>,
152     system_program: &AccountInfo<'a>,
153     data: SignedMetadata,
154     seeds: [u8; 32],
155 ) -> ProgramResult {
156     let create_metadata_instruction = create_metadata_accounts_v2(
157         mpl_token_metadata::id(),
158         *metadata_account.key,
159         *mint.key,
160         *mint_authority.key,
161         *payer.key,
162         *mint_authority.key,
163         data.name,
164         data.symbol,
```

```
        data.uri,
        None,
        0,
        true,
        true,
        None,
        None,
    );

```

Listing 2: src/processor.rs (Line 1006)

```
fn call_create_associated_account<'a>(
    payer: &AccountInfo<'a>,
    wallet: &AccountInfo<'a>,
    mint: &AccountInfo<'a>,
    account: &AccountInfo<'a>,
    rent_info: &AccountInfo<'a>,
    system_program: &AccountInfo<'a>,
    spl_token: &AccountInfo<'a>,
) -> ProgramResult {
    invoke(
        &create_associated_token_account(
            payer.key,
            wallet.key,
            mint.key,
        ),
        &[
            payer.clone(),
            account.clone(),
            wallet.clone(),
            mint.clone(),
            system_program.clone(),
            spl_token.clone(),
            rent_info.clone()
        ],
    )
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Deprecated functions should be replaced according to the following:

- mpl_token_metadata::instruction::create_metadata_accounts_v2 should be replaced by mpl_token_metadata::instruction::create_metadata_accounts_v3
- spl_associated_token_account::create_associated_token_account should be replaced by spl_associated_token_account::instruction::create_associated_token_account

Remediation Plan:

RISK ACCEPTED The Rarimo team accepted the risk and stated that they are not currently considering removing the above functions. They will continue to follow updates to Metaplex and Associated Token Account smart contracts to track that those functions are still usable and do not introduce vulnerabilities in the bridge.

3.2 (HAL-02) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL

Description:

It was observed that there is no overflow-checks=true in Cargo.toml. By default, overflow checks are disabled in optimized release builds. Hence, if there is an overflow on release builds, it will be silenced, leading to unexpected behavior of an application. Even if checked arithmetic is used through checked_*, it is recommended to have that check in Cargo.toml.

Code Location:

- solana-rarimo-bridge-program/Cargo.toml

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation Plan:

SOLVED The Rarimo team resolved this issue in the following commit:
- a446fb7575b2c925e08b22e23b5b7c8adb3db0a4

Overflow checks are now present in the release profile.

3.3 (HAL-03) INTEGER UNDERFLOW - INFORMATIONAL

Description:

An underflow or overflow happens when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented with a given number of bits. If it is not, in Rust the resulting value is specified to wrap as two's complement, resulting in a value either too low or too high considering the circumstances.

Code Location:

Listing 3: src/processor.rs (Line 565)

```
558 if bridge_associated.amount < amount {
559     msg!("Minting token to bridge admin");
560     call_mint_to(
561         mint_info,
562         bridge_associated_info,
563         bridge_admin_info,
564         seeds,
565         amount - bridge_associated.amount,
566     )?;
567 }
```

Listing 4: src/processor.rs (Lines 422-423)

```
421 msg!("Transferring token");
422 **bridge_admin_info.try_borrow_mut_lamports()? -= amount;
423 **owner_info.try_borrow_mut_lamports()? += amount;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using checked_* or saturating_* arithmetic operations instead of regular arithmetic operators to handle overflows gracefully.

Remediation Plan:

SOLVED The Rarimo team solved this issue in the following commit:
- a446fb7575b2c925e08b22e23b5b7c8adb3db0a4

Integer underflow and overflow are now flagged in release builds.

3.4 (HAL-04) DEAD CODE - INFORMATIONAL

Description:

The Bridge program contains dead code, which can never be executed. Jump statements (return, break and continue) and throw expressions move control flow out of the current code block. So, any statements that come after a jump are dead code. Existence of unreachable code decreases readability of the code-base and unnecessary increases program size.

Code Location:

Listing 5: src/processor.rs (Line 1105)

```
1105 fn call_create_master_edition<'a>(
1106     edition: &AccountInfo<'a>,
1107     mint: &AccountInfo<'a>,
1108     update_authority: &AccountInfo<'a>,
1109     mint_authority: &AccountInfo<'a>,
1110     metadata: &AccountInfo<'a>,
1111     payer: &AccountInfo<'a>,
1112     token_program: &AccountInfo<'a>,
1113     system_program: &AccountInfo<'a>,
1114     rent: &AccountInfo<'a>,
1115     seeds: [u8; 32],
1116 ) -> ProgramResult {
1117     let create_master_edition_instruction =
  ↳ create_master_edition_v3(
1118         mpl_token_metadata::id(),
1119         *edition.key,
1120         *mint.key,
1121         *update_authority.key,
1122         *mint_authority.key,
1123         *metadata.key,
1124         *payer.key,
1125         Some(0),
1126     );
1127
1128     invoke_signed(
```

```
129         &create_master_edition_instruction,
130         &[
131             edition.clone(),
132             mint.clone(),
133             update_authority.clone(),
134             mint_authority.clone(),
135             payer.clone(),
136             metadata.clone(),
137             token_program.clone(),
138             system_program.clone(),
139             rent.clone(),
140         ],
141         &[&[&seeds]],
142     )
143 }
```

Listing 6: src/merkle.rs (Line 9)

```
8 const SOLANA_NETWORK: &str = "Solana";
9 const SOLANA_NATIVE_DECIMALS: u8 = 9u8;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider deleting all blocks of code that do not implement program business logic in any way. Doing so decreases the cost of deploying the program.

Remediation Plan:

SOLVED The Rarimo team resolved this issue in the following commit:
- a1a4f680213bcf0e6baa257f9ef4ea62da870448

All dead code was removed.

3.5 (HAL-05) LACK OF TEST COVERAGE - INFORMATIONAL

Description:

Checking the code by automated testing (unit testing or functional testing) is a good practice to ensure that all lines of the code work correctly. It was observed that no tests were present for any of the existing instructions and program logic.

Code Location:

- solana-rarimo-bridge-program/tests

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to implement as many test cases as possible to cover all scenarios in the program flow.

Remediation Plan:

ACKNOWLEDGED The Rarimo team acknowledged this issue and stated that they will consider adding tests for separate methods in a later release. They also have some tests written in Golang in their private development repository.

3.6 (HAL-06) POSSIBLE RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL

Description:

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

Code Location:

Listing 7

```
   src/processor.rs:385:    let bridge_admin_key = Pubkey::
↳ create_program_address(&[&seeds], &program_id).unwrap();
   src/processor.rs:478:    let bridge_admin_key = Pubkey::
↳ create_program_address(&[&seeds], &program_id).unwrap();
   src/processor.rs:646:    let bridge_admin_key = Pubkey::
↳ create_program_address(&[&seeds], &program_id).unwrap();
   src/processor.rs:680:            Some(metadata.collection.unwrap().
↳ key.to_bytes())
   src/processor.rs:814:    let bridge_admin_key = Pubkey::
↳ create_program_address(&[&seeds], &program_id).unwrap();
   src/processor.rs:911:        Ok::<SignedMetadata, BridgeError>(
↳ signed_meta.unwrap())
   src/util.rs:19:    if recovered_key.unwrap().0 != target_key {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to use the unwrap function in the production environment because its use causes panic! and may crash any affected module, program or, in the worst case, the runtime without verbose error messages. Crashing the system will result in a loss of availability and, in some cases, even private information stored in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrapping, or using the error-chain crate for errors.

Remediation Plan:

SOLVED The Rarimo team resolved this issue in the following commits:
- a1a4f680213bcf0e6baa257f9ef4ea62da870448
- a446fb7575b2c925e08b22e23b5b7c8adb3db0a4

All unsafe unwraps were removed.

MANUAL TESTING

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

4.1 UNAUTHORIZED WITHDRAWAL

Description:

It was attempted to bypass the implemented flow to withdraw funds that were not previously deposited into the bridge. Withdrawals require a valid Merkle node, Merkle path, an admin signature for the Merkle root and a valid ECDSA signature. Several tests were performed to ensure that it was not possible to bypass the Merkle validations, regarding the ECDSA signature, since it was using the native `secp256k1_recover` program it was ensured that it was used according to best practices.

Results:

No code vulnerabilities were identified.

4.2 BRIDGE ADMIN MODIFICATION

Description:

Since the `BridgeAdmin` Program Derived Address (PDA) is used for storing tokens, either natively or through an associated token account, it was checked if it was possible to misuse the `TransferOwnership` to change the stored ECDSA public key. The instruction requires the `BridgeAdmin` account before changing the public key it verifies with `secp256k1_recover` that the old public key, signature, and the recovery ID match.

Results:

No code vulnerabilities were identified.

4.3 LOCKED FUNDS

Description:

The Bridge provides a set of instructions for depositing and withdrawing native SOL tokens, Fungible Tokens and Non-Fungible Tokens. Since being able to withdraw previously deposited tokens in another chain is essential, several checks such as attempting the same withdrawal multiple times, passing wrong accounts, seeds and signatures, to ensure whether it was possible to lock tokens in the protocol were performed.

Results:

No code vulnerabilities were identified.

AUTOMATED TESTING

5.1 AUTOMATED VULNERABILITY SCANNING

Description:

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

Results:

Soteria reported three issues related to unsafe math, which are already covered in HAL-03.

AUTOMATED TESTING

```
anchor_lang_version: 3.5.0 anchorVersionTooOld: 0
 - ✔ [00m:01s] Loading IR From File
 - ▪ [00m:00s] Running Compiler Optimization Passes
EntryPoints:
entrypoint
 - ✔ [00m:00s] Running Compiler Optimization Passes
 - ✔ [00m:01s] Running Pointer Analysis

=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 565, column 13 in src/processor.rs
The sub operation may result in underflows:

 559|        msg!("Minting token to bridge admin");
 560|        call_mint_to(
 561|            mint_info,
 562|            bridge_associated_info,
 563|            bridge_admin_info,
 564|            seeds,
>565|            amount - bridge_associated.amount,
 566|        )?;
 567|    }
 568|
 569|    msg!("Transferring token");
 570|    call_transfer_token(
 571|        bridge_associated_info,
>>>Stack Trace:
>>>bridge::processor::process_instruction::hb98accbe680d075f [src/entrypoint.rs:21]
>>>  bridge::processor::process_withdraw_ft::h025919651857e586 [src/processor.rs:75]

=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 422, column 5 in src/processor.rs
The sub operation may result in underflows:

 416|        WITHDRAW_SIZE,
 417|        program_id,
 418|        &[origin.as_slice(), &[bump_seed]],
 419|    )?;
 420|
 421|    msg!("Transferring token");
>422|    **bridge_admin_info.try_borrow_mut_lamports()? -= amount;
 423|    **owner_info.try_borrow_mut_lamports()? += amount;
 424|
 425|    msg!("Initializing withdraw account");
 426|    let mut withdraw: Withdraw = BorshDeserialize::deserialize(&mut withdraw_info.data.borrow_mut().as_ref())?;
 427|    if withdraw.is_initialized {
 428|        return Err(BridgeError::AlreadyInUse.into());
>>>Stack Trace:
>>>bridge::processor::process_instruction::hb98accbe680d075f [src/entrypoint.rs:21]
>>>  bridge::processor::process_withdraw_native::h7ab873cfc421c8d7 [src/processor.rs:69]
```

```
=============This arithmetic operation may be UNSAFE!================
Found a potential vulnerability at line 423, column 5 in src/processor.rs
The add operation may result in overflows:

 417|        program_id,
 418|        &[origin.as_slice(), &[bump_seed]],
 419|    )?;
 420|
 421|    msg!("Transferring token");
 422|    **bridge_admin_info.try_borrow_mut_lamports()? -= amount;
>423|    **owner_info.try_borrow_mut_lamports()? += amount;
 424|
 425|    msg!("Initializing withdraw account");
 426|    let mut withdraw: Withdraw = BorshDeserialize::deserialize(&mut withdraw_info.data.borrow_mut().as_ref())?;
 427|    if withdraw.is_initialized {
 428|        return Err(BridgeError::AlreadyInUse.into());
 429|    }
>>>Stack Trace:
>>>bridge::processor::process_instruction::hb98accbe680d075f [src/entrypoint.rs:21]
>>>  bridge::processor::process_withdraw_native::h7ab873cfc421c8d7 [src/processor.rs:69]

 - ✔ [00m:00s] Building Static Happens-Before Graph
 - ✔ [00m:00s] Detecting Vulnerabilities
detected 3 unsafe math operations in total.

--------The summary of potential vulnerabilities in all.ll--------

        3 unsafe arithmetic issues
```

5.2 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Results:

No vulnerabilities were identified.

5.3 UNSAFE RUST CODE DETECTION

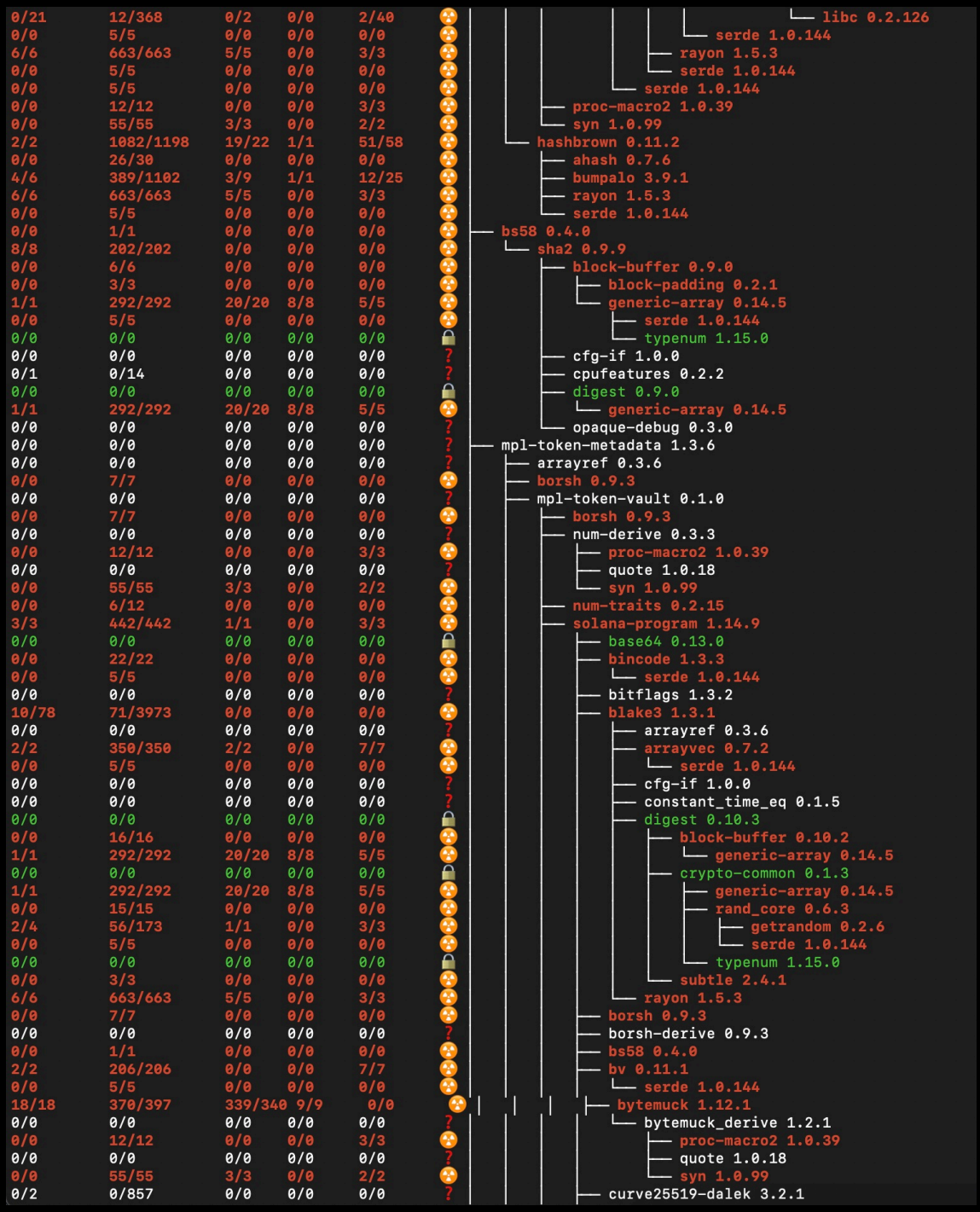
Description:

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

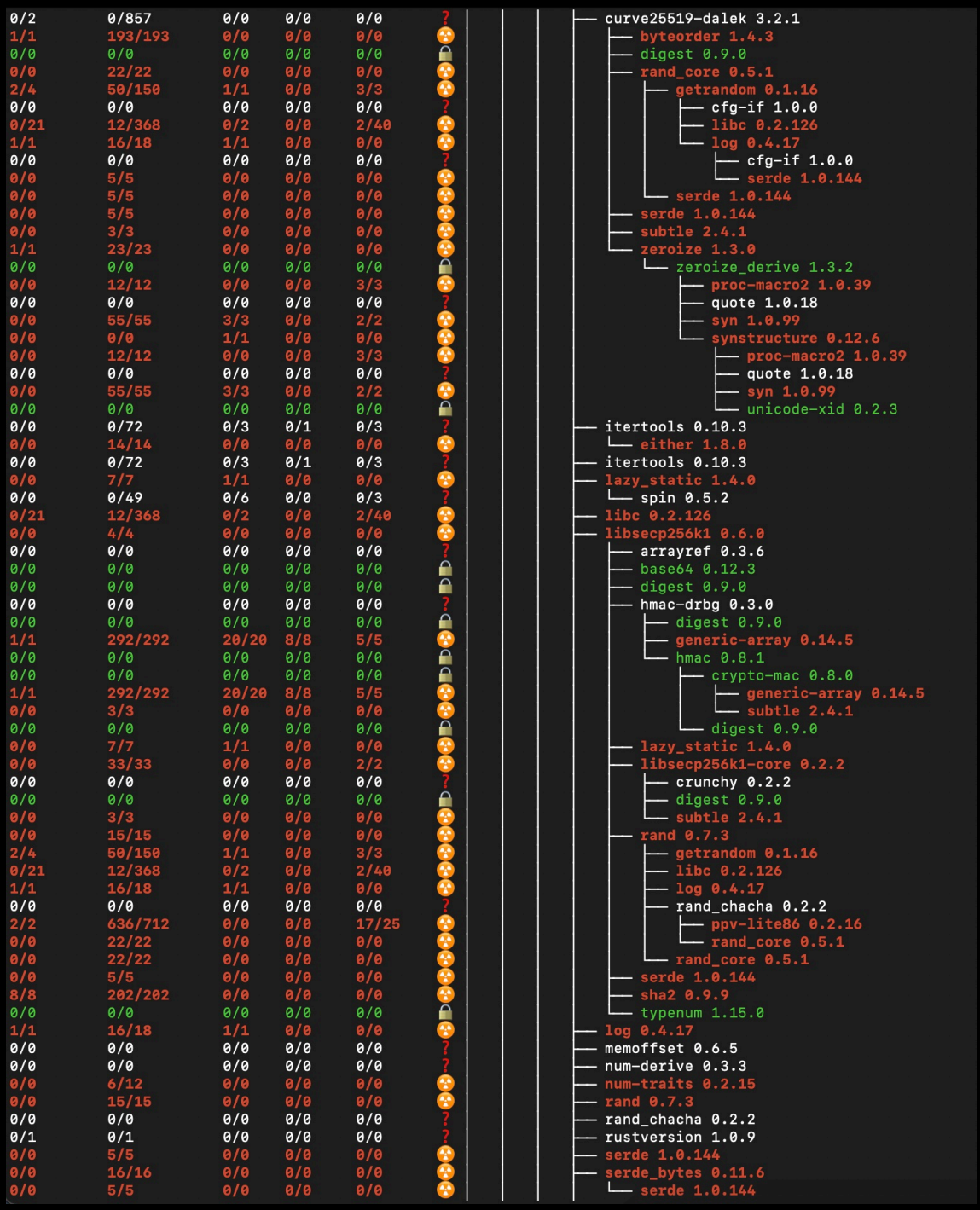
Results:

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

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ solana-bridge-program 0.0.1
0/0        0/0          0/0    0/0     0/0      ❓ ├── arrayref 0.3.6
0/0        7/7          0/0    0/0     0/0      ☢️ ├── borsh 0.9.3
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── borsh-derive 0.9.3
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── borsh-derive-internal 0.9.3
0/0        12/12        0/0    0/0     3/3      ☢️ │   │   │   ├── proc-macro2 1.0.39
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │   │   └── unicode-ident 1.0.0
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   ├── quote 1.0.18
0/0        12/12        0/0    0/0     3/3      ☢️ │   │   │   │   └── proc-macro2 1.0.39
0/0        55/55        3/3    0/0     2/2      ☢️ │   │   │   └── syn 1.0.99
0/0        12/12        0/0    0/0     3/3      ☢️ │   │   │       ├── proc-macro2 1.0.39
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       ├── quote 1.0.18
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │       └── unicode-ident 1.0.0
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── borsh-schema-derive-internal 0.9.3
0/0        12/12        0/0    0/0     3/3      ☢️ │   │   │   ├── proc-macro2 1.0.39
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   ├── quote 1.0.18
0/0        55/55        3/3    0/0     2/2      ☢️ │   │   │   └── syn 1.0.99
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── proc-macro-crate 0.1.5
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │   └── toml 0.5.9
0/0        0/46         0/1    0/0     0/0      ❓ │   │   │       ├── indexmap 1.9.2
1/1        1241/1367    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        56/173       1/1    0/0     3/3      ☢️ │   │   │       │   │   │   ├── getrandom 0.2.6
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │   ├── cfg-if 1.0.0
0/21       12/368       0/2    0/0     2/40     ☢️ │   │   │       │   │   │   │   └── libc 0.2.126
1/1        76/118       4/6    0/0     2/3      ☢️ │   │   │       │   │   │   ├── once_cell 1.13.1
0/16       0/1341       0/0    0/0     0/56     ❓ │   │   │       │   │   │   │   └── parking_lot_core 0.9.3
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │       ├── cfg-if 1.0.0
0/21       12/368       0/2    0/0     2/40     ☢️ │   │   │       │   │   │   │       ├── libc 0.2.126
0/1        0/392        0/7    0/1     0/13     ❓ │   │   │       │   │   │   │       └── smallvec 1.8.0
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │       │   │   │   │           └── serde 1.0.144
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │               └── serde_derive 1.0.144
0/0        12/12        0/0    0/0     3/3      ☢️ │   │   │       │   │   │   │                   ├── proc-macro2 1.0.39
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │                   ├── quote 1.0.18
0/0        55/55        3/3    0/0     2/2      ☢️ │   │   │       │   │   │   │                   └── syn 1.0.99
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │       │   │   │   └── serde 1.0.144
4/6        389/1102     3/9    1/1     12/25    ☢️ │   │   │       │   │   ├── bumpalo 3.9.1
6/6        663/663      5/5    0/0     3/3      ☢️ │   │   │       │   │   ├── rayon 1.5.3
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        435/447      9/9    0/0     28/28    ☢️ │   │   │       │   │   │   │   ├── crossbeam-epoch 0.9.10
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │   │   ├── cfg-if 1.0.0
4/4        81/81        14/14  0/0     2/2      ☢️ │   │   │       │   │   │   │   │   ├── crossbeam-utils 0.8.11
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │   │   │   ├── cfg-if 1.0.0
1/1        76/118       4/6    0/0     2/3      ☢️ │   │   │       │   │   │   │   │   │   └── once_cell 1.13.1
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       │   │   │   │   │   ├── memoffset 0.6.5
1/1        76/118       4/6    0/0     2/3      ☢️ │   │   │       │   │   │   │   │   ├── once_cell 1.13.1
0/0        18/18        1/1    0/0     0/0      ☢️ │   │   │       │   │   │   │   │   └── scopeguard 1.1.0
4/4        81/81        14/14  0/0     2/2      ☢️ │   │   │       │   │   │   │   └── crossbeam-utils 0.8.11
0/0        14/14        0/0    0/0     0/0      ☢️ │   │   │       │   │   │   ├── either 1.8.0
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │       │   │   │   │   └── serde 1.0.144
5/5        485/488      2/2    0/0     20/20    ☢️ │   │   │       │   │   │   └── rayon-core 1.9.3
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        81/81        14/14  0/0     2/2      ☢️ │   │   │       │   │   │       │   └── crossbeam-utils 0.8.11
0/0        453/453      6/6    0/0     6/6      ☢️ │   │   │       │   │   │       ├── crossbeam-deque 0.8.2
4/4        81/81        14/14  0/0     2/2      ☢️ │   │   │       │   │   │       ├── crossbeam-utils 0.8.11
0/0        72/72        0/0    0/0     0/0      ☢️ │   │   │       │   │   │       └── num_cpus 1.13.1
0/21       12/368       0/2    0/0     2/40     ☢️ │   │   │       │   │   │           └── libc 0.2.126
```

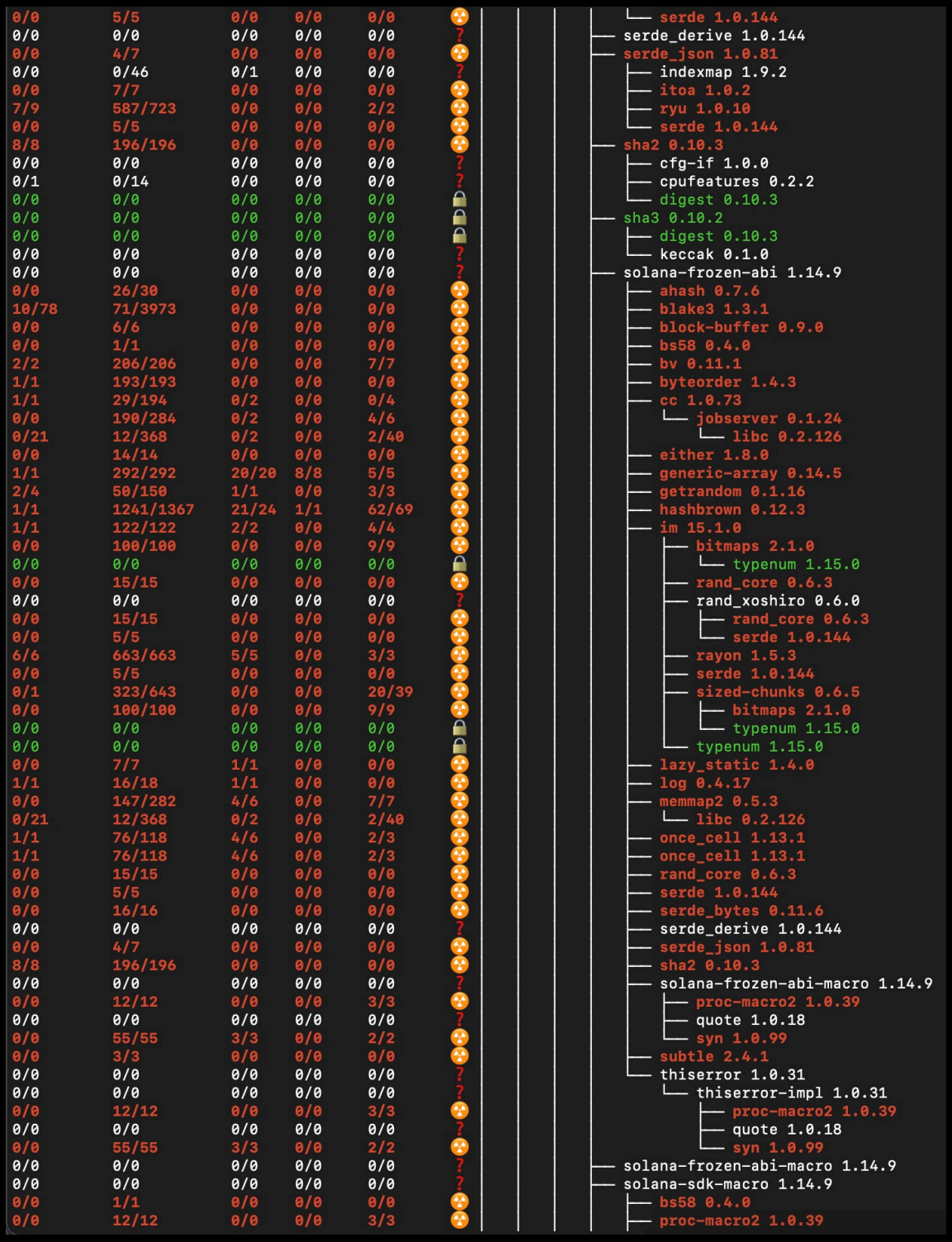
AUTOMATED TESTING



AUTOMATED TESTING

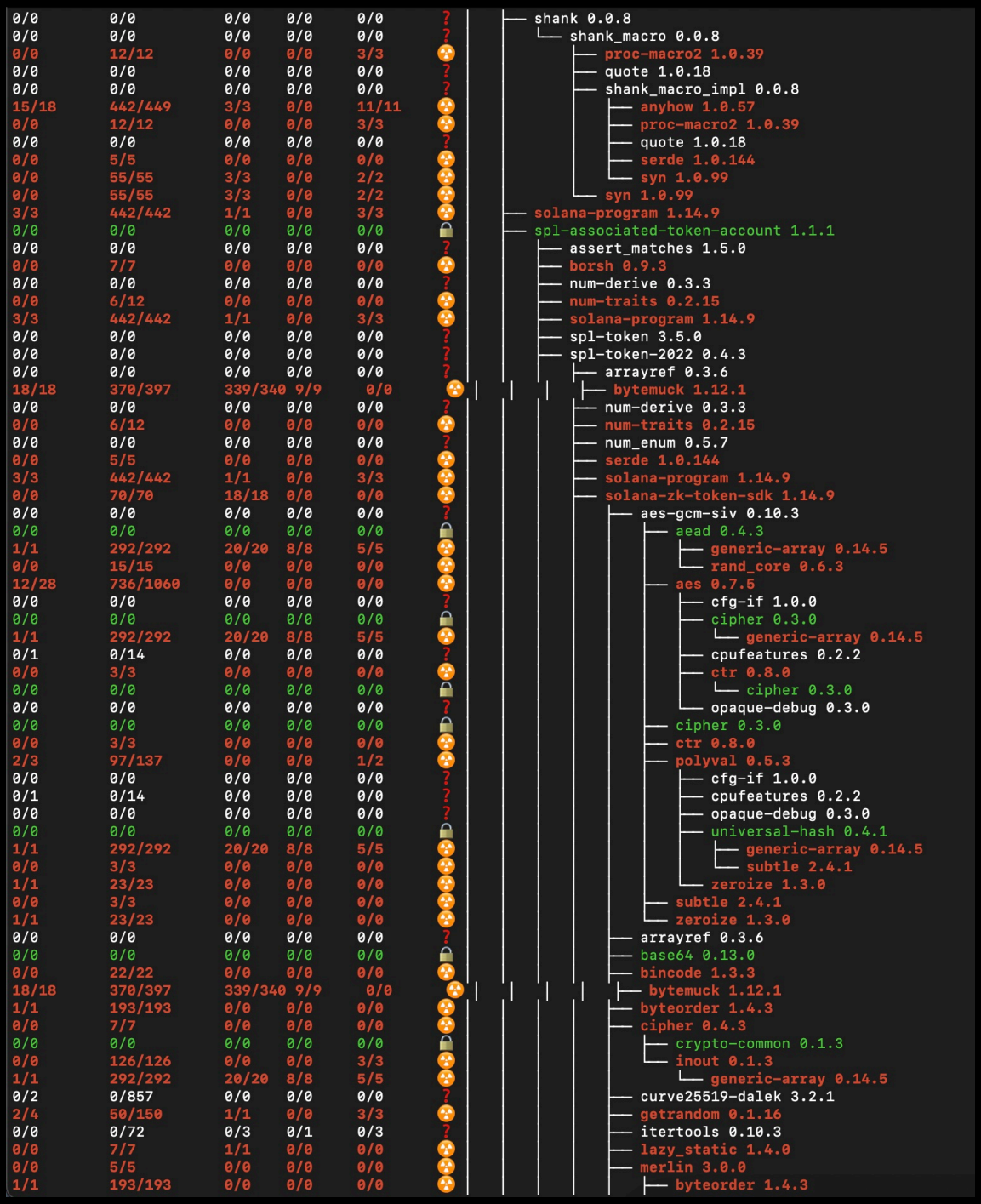


AUTOMATED TESTING

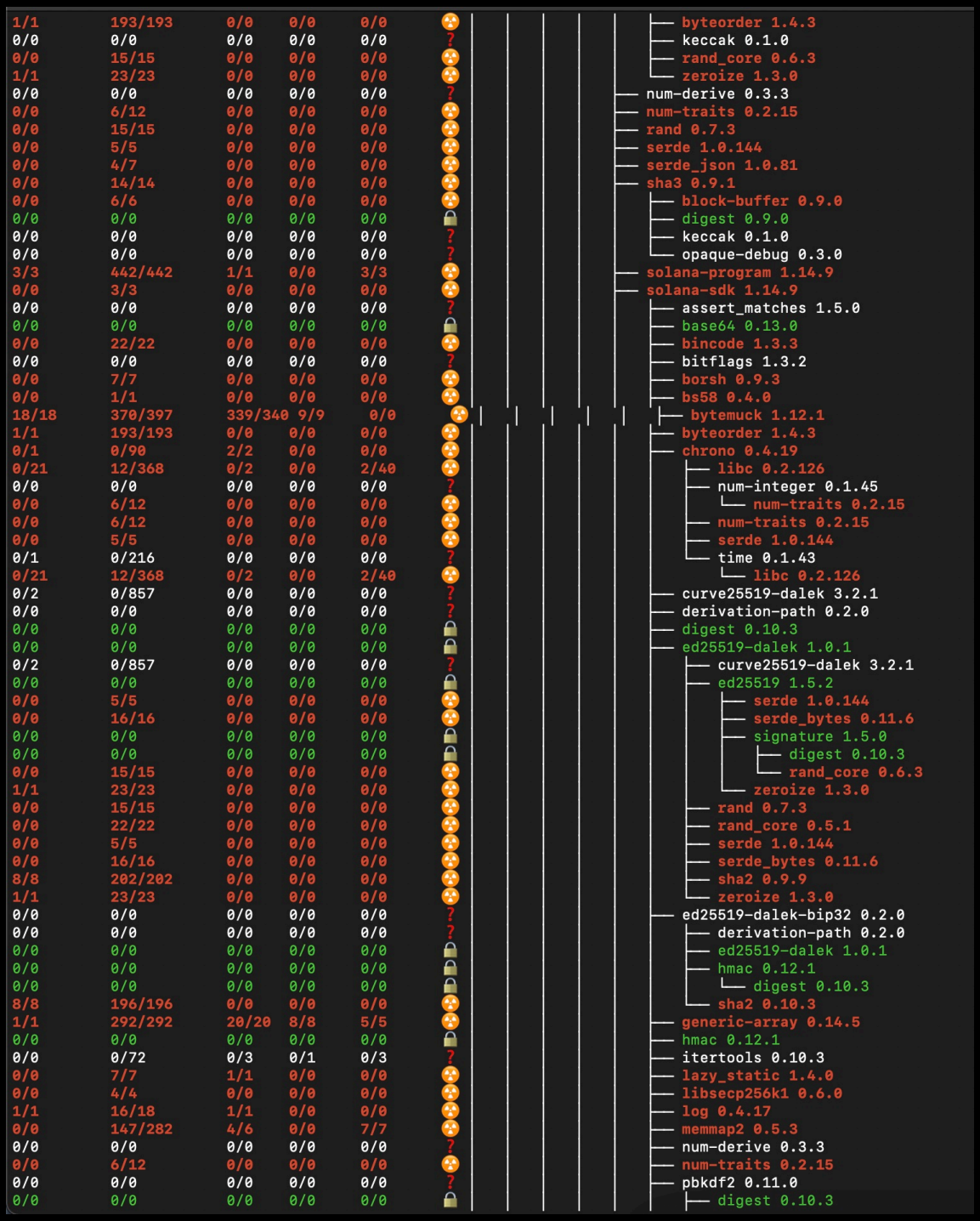


AUTOMATED TESTING

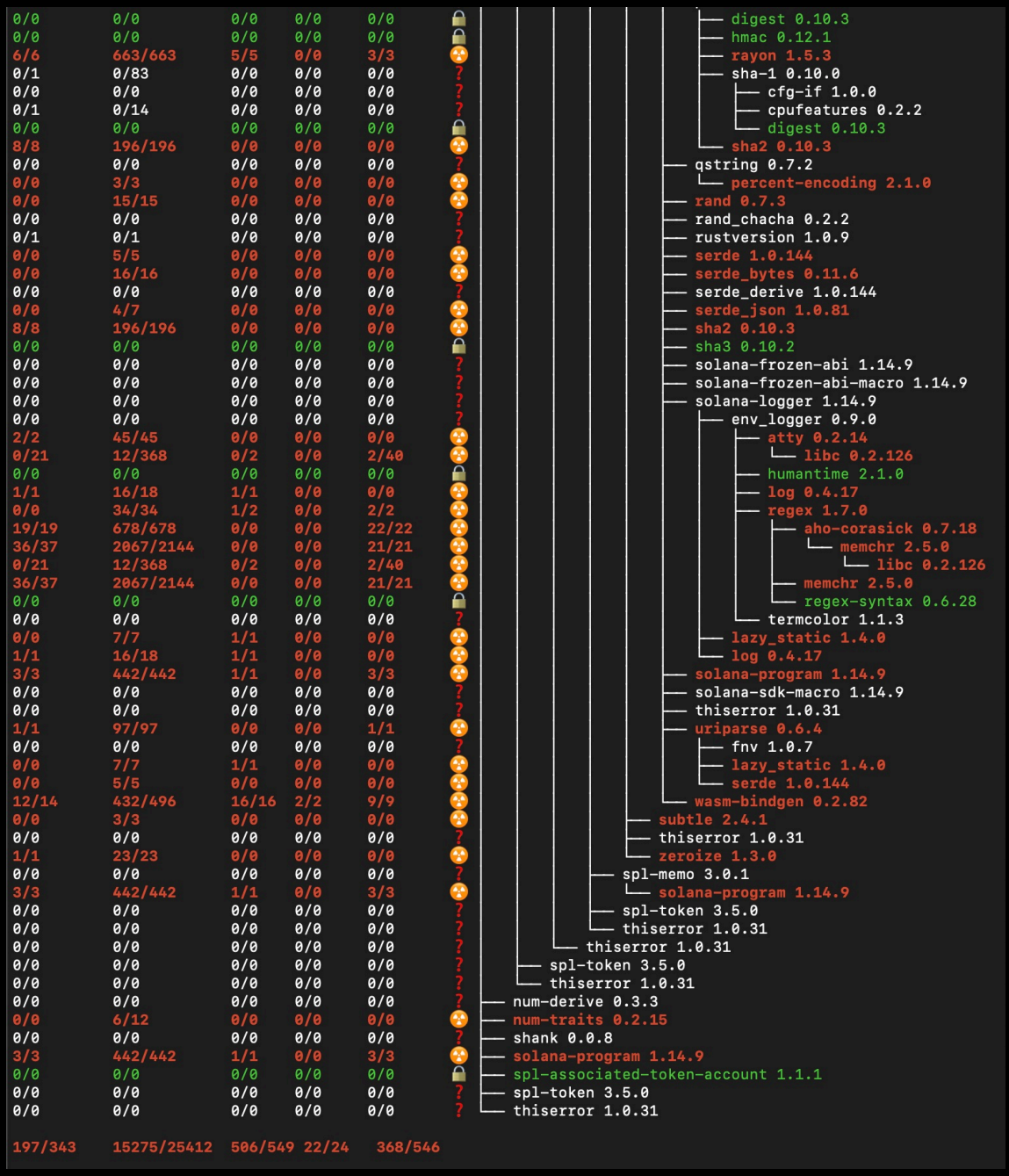
AUTOMATED TESTING



AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN