Solana Labs - SPL Token 2022

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 27th, 2022 - August 19th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/19/2022	Michael Smith
0.2	Final Draft	08/19/2022	Michael Smith
0.3	Draft Review	08/22/2022	Piotr Cielas
0.4	Draft Review	08/22/2022	Gabi Urrutia
1.0	Remediation Plan	11/22/2022	Michael Smith
1.1	Remediation Plan Review	11/24/2022	Isabel Burruezo
1.2	Remediation Plan Review	11/24/2022	Piotr Cielas
1.3	Remediation Plan Review	11/24/2022	Gabi Urrutia

CONTACTS

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Solana Labs engaged Halborn to conduct a security audit on their Solana programs beginning on July 27th, 2022 and ending on August 19th, 2022. The security assessment was scoped to the program provided in the solana-program-library GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

SPL Token 2022 is a superset of Solana Lab's original Token program, defining most of the needs for fungible and non-fungible tokens. It provides additional functionality through an extension model, allowing developers to customize their tokens according to their needs.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

In summary, Halborn identified some security risks that were mostly addressed by the Solana Labs team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the 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 logic, process, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

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

- Research into the architecture, purpose, and use of the platform.
- Solana program manual code review and walkthrough to identify any logic issue.
- 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:

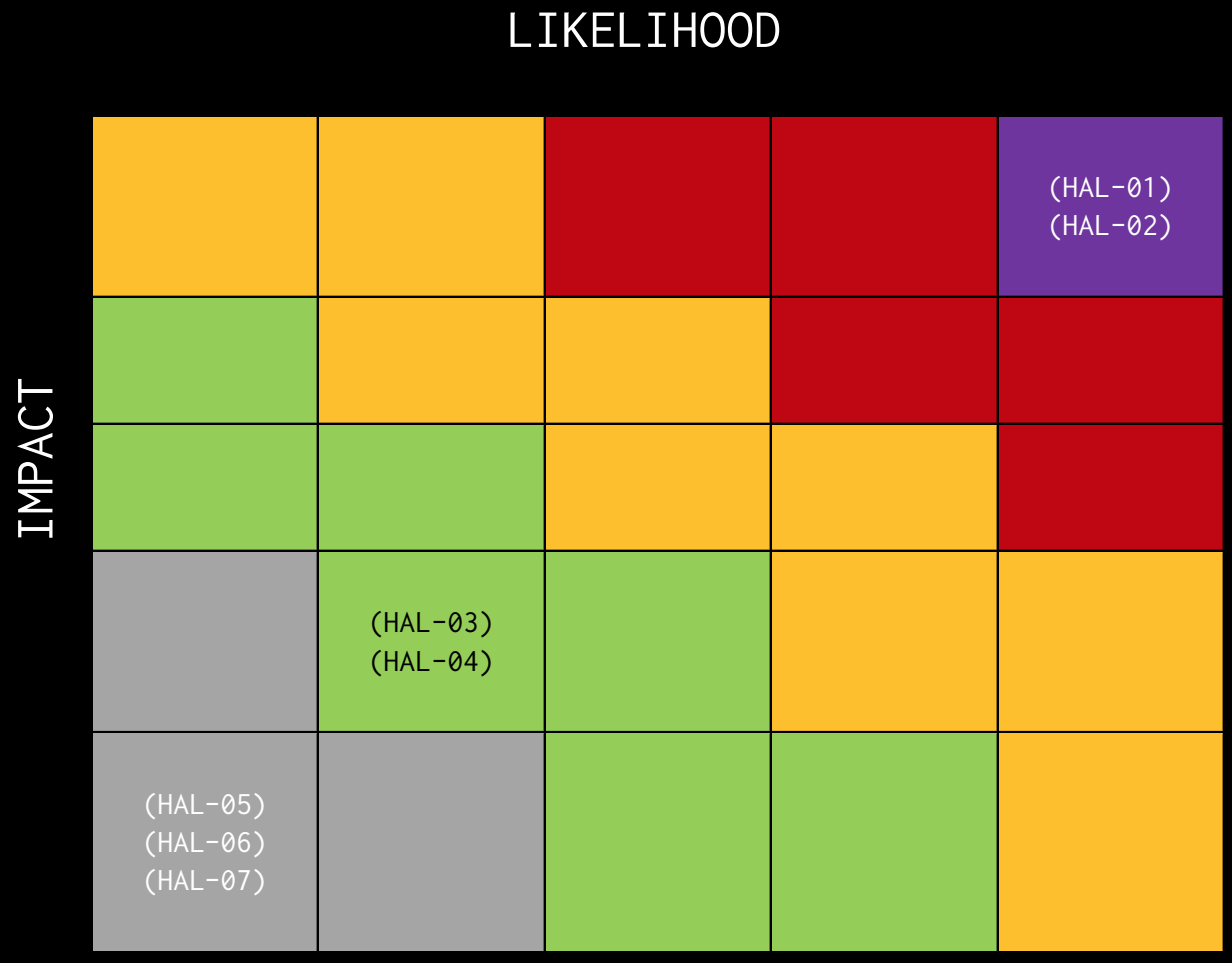
1. SPL Token 2022

- Repository: solana-program-library
- Commit ID: c3137af9dfa2cc0873cc84c4418dea88ac542965
- Programs in scope:
    1. (token/program-2022)

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	0	0	2	3



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) USERS CAN CIRCUMVENT TRANSFER FEE	Critical	SOLVED - 11/16/2022
(HAL-02) NON-TRANSFERABLE TOKENS CAN BE TRANSFERRED	Critical	SOLVED - 11/16/2022
(HAL-03) MEMO REQUIREMENT CAN BE BYPASSED	Low	SOLVED - 11/16/2022
(HAL-04) AMOUNT CASTING - IMPRECISION/OVERFLOW	Informational	ACKNOWLEDGED
(HAL-05) MISSING CARGO OVERFLOW CHECKS	Informational	ACKNOWLEDGED
(HAL-06) ZERO AMOUNT PROCESSING	Informational	FUTURE RELEASE
(HAL-07) MINT INITIALIZATION FLOW	Informational	SOLVED - 11/16/202

FINDINGS & TECH DETAILS

3.1 (HAL-01) USERS CAN CIRCUMVENT TRANSFER FEE - CRITICAL

Description:

The Token-2022 program introduces a transfer fee extension, allowing developers to charge users a fee when transferring tokens. However, users can deposit tokens from their token account into their Confidential Token Account and withdraw tokens from their Confidential Token Account into their Token Account without paying the transfer fee. This is due to the withdraw and deposit instructions allowing users to specify which account to deposit/withdraw the tokens to, thus the withdraw and deposit instructions can be used as a pseudo transfer instruction.

To reproduce:

Scenario #1

1. Create a mint with the transfer fee extension enabled
2. Create two Token Accounts (Alice and Bob)
3. Deposit tokens into Alice's Confidential Token account.
4. Withdraw the tokens from Alice's Confidential Token Account into Bob's Token Account

Scenario #2

1. Create a mint with the transfer fee extension enabled
2. Create two Token Accounts (Alice and Bob)
3. Deposit tokens into Bob's Token accounts.
4. Deposit the tokens from Bob's Token Account into Alice's Confidential Token Account.

Notice that the credited Token Account has all the transferred tokens, as no fee was removed as required by the transfer fee extension.

Code Location:

Listing 1: src/extension/confidential_transfer/processor.rs (Lines 269,345-370)

```
261 fn process_deposit(
262     program_id: &Pubkey,
263     accounts: &[AccountInfo],
264     amount: u64,
265     expected_decimals: u8,
266 ) -> ProgramResult {
267     let account_info_iter = &mut accounts.iter();
268     let token_account_info = next_account_info(account_info_iter)
 ↳ ?;
269     let destination_token_account_info = next_account_info(
 ↳ account_info_iter)?;
270     let mint_info = next_account_info(account_info_iter)?;
271     let authority_info = next_account_info(account_info_iter)?;
272     let authority_info_data_len = authority_info.data_len();
273
274     check_program_account(mint_info.owner)?;
275     let mint_data = &mint_info.data.borrow_mut();
276     let mint = StateWithExtensions::<Mint>::unpack(mint_data)?;
277
278     if expected_decimals != mint.base.decimals {
279         return Err(TokenError::MintDecimalsMismatch.into());
280     }
281
282     // Process source account
283     {
284         check_program_account(token_account_info.owner)?;
285         let token_account_data = &mut token_account_info.data.
 ↳ borrow_mut();
286         let mut token_account = StateWithExtensionsMut::<Account
 ↳ >::unpack(token_account_data)?;
287
288         Processor::validate_owner(
289             program_id,
290             &token_account.base.owner,
291             authority_info,
292             authority_info_data_len,
293             account_info_iter.as_slice(),
294         )?;
295
```

```
296         if token_account.base.is_frozen() {
297             return Err(TokenError::AccountFrozen.into());
298         }
299
300         if token_account.base.mint != *mint_info.key {
301             return Err(TokenError::MintMismatch.into());
302         }
303
304         // Wrapped SOL deposits are not supported because lamports
 ↳  cannot be vanished.
305         assert!(!token_account.base.is_native());
306         token_account.base.amount = token_account
307             .base
308             .amount
309             .checked_sub(amount)
310             .ok_or(TokenError::Overflow)?;
311
312         token_account.pack_base();
313     }
314
315     //
316     // Finished with the source token account at this point. Drop
 ↳ all references to it to avoid a
317     // double borrow if the source and destination accounts are
 ↳ the same
318     //
319
320     // Process destination account
321     {
322         check_program_account(destination_token_account_info.owner
 ↳ )?;
323         let destination_token_account_data = &mut
 ↳ destination_token_account_info.data.borrow_mut();
324         let mut destination_token_account =
325             StateWithExtensionsMut::<Account>::unpack(
 ↳ destination_token_account_data)?;
326
327         if destination_token_account.base.is_frozen() {
328             return Err(TokenError::AccountFrozen.into());
329         }
330
331         if destination_token_account.base.mint != *mint_info.key {
332             return Err(TokenError::MintMismatch.into());
333         }
```

```
334
335         let mut destination_confidential_transfer_account =
336             destination_token_account.get_extension_mut::<
 ↳ ConfidentialTransferAccount>()?;
337         destination_confidential_transfer_account.approved()?;
338
339         if !bool::from(&destination_confidential_transfer_account.
 ↳ allow_balance_credits) {
340             return Err(TokenError::
 ↳ ConfidentialTransferDepositsAndTransfersDisabled.into());
341         }
342
343         // Divide deposit into the low 16 and high 48 bits and
 ↳ then add to the appropriate pending
344         // ciphertexts
345         destination_confidential_transfer_account.
 ↳ pending_balance_lo = ops::add_to(
346             &destination_confidential_transfer_account.
 ↳ pending_balance_lo,
347             amount << PENDING_BALANCE_HI_BIT_LENGTH >>
 ↳ PENDING_BALANCE_HI_BIT_LENGTH,
348         )
349         .ok_or(ProgramError::InvalidInstructionData)?;
350
351         destination_confidential_transfer_account.
 ↳ pending_balance_hi = ops::add_to(
352             &destination_confidential_transfer_account.
 ↳ pending_balance_hi,
353             amount >> PENDING_BALANCE_LO_BIT_LENGTH,
354         )
355         .ok_or(ProgramError::InvalidInstructionData)?;
356
357         destination_confidential_transfer_account.
 ↳ pending_balance_credit_counter =
358             (u64::from(destination_confidential_transfer_account.
 ↳ pending_balance_credit_counter)
359                 .checked_add(1)
360                 .ok_or(ProgramError::InvalidInstructionData)?)
361             .into();
362
363         if u64::from(destination_confidential_transfer_account.
 ↳ pending_balance_credit_counter)
364             > u64::from(
365                 destination_confidential_transfer_account.
```

```
 ↳ maximum_pending_balance_credit_counter,
366             )
367         {
368             return Err(TokenError::
 ↳ MaximumPendingBalanceCreditCounterExceeded.into());
369         }
370     }
```

Listing 2: src/extension/confidential_transfer/processor.rs (Lines 387,453-474)

```
377     fn process_withdraw(
378      program_id: &Pubkey,
379      accounts: &[AccountInfo],
380      amount: u64,
381      expected_decimals: u8,
382      new_decryptable_available_balance: DecryptableBalance,
383      proof_instruction_offset: i64,
384 ) -> ProgramResult {
385      let account_info_iter = &mut accounts.iter();
386      let token_account_info = next_account_info(account_info_iter)
 ↳ ?;
387      let destination_token_account_info = next_account_info(
 ↳ account_info_iter)?;
388      let mint_info = next_account_info(account_info_iter)?;
389      let instructions_sysvar_info = next_account_info(
 ↳ account_info_iter)?;
390      let authority_info = next_account_info(account_info_iter)?;
391      let authority_info_data_len = authority_info.data_len();
392
393      check_program_account(mint_info.owner)?;
394      let mint_data = &mint_info.data.borrow_mut();
395      let mint = StateWithExtensions::<Mint>::unpack(mint_data)?;
396
397      if expected_decimals != mint.base.decimals {
398          return Err(TokenError::MintDecimalsMismatch.into());
399      }
400
401      let previous_instruction =
402          get_instruction_relative(proof_instruction_offset,
 ↳ instructions_sysvar_info)?;
403
404      let proof_data = decode_proof_instruction::<WithdrawData>(
```

```
405         ProofInstruction::VerifyWithdraw,
406         &previous_instruction,
407     )?;
408
409     // Process source account
410     {
411         check_program_account(token_account_info.owner)?;
412         let token_account_data = &mut token_account_info.data.
 ↳ borrow_mut();
413         let mut token_account = StateWithExtensionsMut::<Account
 ↳ >::unpack(token_account_data)?;
414
415         Processor::validate_owner(
416             program_id,
417             &token_account.base.owner,
418             authority_info,
419             authority_info_data_len,
420             account_info_iter.as_slice(),
421         )?;
422
423         if token_account.base.is_frozen() {
424             return Err(TokenError::AccountFrozen.into());
425         }
426
427         if token_account.base.mint != *mint_info.key {
428             return Err(TokenError::MintMismatch.into());
429         }
430
431         let mut confidential_transfer_account =
432             token_account.get_extension_mut::<
 ↳ ConfidentialTransferAccount>()?;
433
434         confidential_transfer_account.available_balance =
435             ops::subtract_from(&confidential_transfer_account.
 ↳ available_balance, amount)
436                 .ok_or(ProgramError::InvalidInstructionData)?;
437
438         if confidential_transfer_account.available_balance !=
 ↳ proof_data.final_ciphertext {
439             return Err(TokenError::
 ↳ ConfidentialTransferBalanceMismatch.into());
440         }
441
442         confidential_transfer_account.
```

```
 ↳ decryptable_available_balance =
443                 new_decryptable_available_balance;
444             }
445
446             //
447             // Finished with the source token account at this point. Drop
 ↳ all references to it to avoid a
448             // double borrow if the source and destination accounts are
 ↳ the same
449             //
450
451             // Process destination account
452             {
453                 check_program_account(destination_token_account_info.owner
 ↳ )?;
454                 let destination_token_account_data = &mut
 ↳ destination_token_account_info.data.borrow_mut();
455                 let mut destination_token_account =
456                     StateWithExtensionsMut::<Account>::unpack(
 ↳ destination_token_account_data)?;
457
458                 if destination_token_account.base.is_frozen() {
459                     return Err(TokenError::AccountFrozen.into());
460                 }
461
462                 if destination_token_account.base.mint != *mint_info.key {
463                     return Err(TokenError::MintMismatch.into());
464                 }
465
466                 // Wrapped SOL withdrawals are not supported because
 ↳ lamports cannot be apparated.
467                 assert!(!destination_token_account.base.is_native());
468                 destination_token_account.base.amount =
 ↳ destination_token_account
469                     .base
470                     .amount
471                     .checked_add(amount)
472                     .ok_or(TokenError::Overflow)?;
473
474                 destination_token_account.pack_base();
475             }
476
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Ensure the source and destination account for the deposit and withdraw function are the same. This will force users to use the transfer instruction to move tokens between accounts.

Remediation Plan:

SOLVED The Solana Labs team resolved this issue in commit 3ddb3c7404d23146a390150e241831e116c5cc8d: The destination_token account has been removed from the withdraw and deposit instruction, deposits & withdraws will be sent to the users Token/Confidential Token Account.

3.2 (HAL-02) NON-TRANSFERABLE TOKENS CAN BE TRANSFERRED - CRITICAL

Description:

The token-2022 program introduces an non-tranferable extension allowing developers to create non-transferable tokens. This can be circumvented if the Confidential_transfer extension is also enabled, users can execute a Confidential Transfer deposit, withdraw or transfer instruction bypassing the non-transferable extension. This is due to the Confidential Transfer extension failing to validate if a mint has the non-transferable extension enabled.

To reproduce:

1. Create a mint with the transfer fee extension enabled
2. Create two Token Accounts (Alice and Bob)
3. Deposit tokens into Alice's Confidential Token Account
4. Confidentially transfer the tokens from Alice's confidential token account to Bob's token account via the withdraw or transfer instructions

Notice that the tokens have been transferred from Alice to Bob.

Code Location:

Listing 3: src/extension/confidential_transfer/processor.rs

```
261 fn process_deposit(
262     program_id: &Pubkey,
263     accounts: &[AccountInfo],
264     amount: u64,
265     expected_decimals: u8,
266 ) -> ProgramResult {
267     let account_info_iter = &mut accounts.iter();
```

```
268     let token_account_info = next_account_info(account_info_iter)
 ↳ ?;
269     let destination_token_account_info = next_account_info(
 ↳ account_info_iter)?;
270     let mint_info = next_account_info(account_info_iter)?;
271     let authority_info = next_account_info(account_info_iter)?;
272     let authority_info_data_len = authority_info.data_len();
273 
274     check_program_account(mint_info.owner)?;
275     let mint_data = &mint_info.data.borrow_mut();
276     let mint = StateWithExtensions::<Mint>::unpack(mint_data)?;
277 
278     if expected_decimals != mint.base.decimals {
279         return Err(TokenError::MintDecimalsMismatch.into());
280     }
281 
282     // Process source account
283     {
284         check_program_account(token_account_info.owner)?;
285         let token_account_data = &mut token_account_info.data.
 ↳ borrow_mut();
286         let mut token_account = StateWithExtensionsMut::<Account
 ↳ >::unpack(token_account_data)?;
287 
288         Processor::validate_owner(
289             program_id,
290             &token_account.base.owner,
291             authority_info,
292             authority_info_data_len,
293             account_info_iter.as_slice(),
294         )?;
295 
296         if token_account.base.is_frozen() {
297             return Err(TokenError::AccountFrozen.into());
298         }
299 
300         if token_account.base.mint != *mint_info.key {
301             return Err(TokenError::MintMismatch.into());
302         }
303 
304         // Wrapped SOL deposits are not supported because lamports
 ↳  cannot be vanished.
305         assert!(!token_account.base.is_native());
306         token_account.base.amount = token_account
```

```
307             .base
308             .amount
309             .checked_sub(amount)
310             .ok_or(TokenError::Overflow)?;
311
312         token_account.pack_base();
313     }
314
315     //
316     // Finished with the source token account at this point. Drop
 ↳ all references to it to avoid a
317     // double borrow if the source and destination accounts are
 ↳ the same
318     //
319
320     // Process destination account
321     {
322         check_program_account(destination_token_account_info.owner
 ↳ )?;
323         let destination_token_account_data = &mut
 ↳ destination_token_account_info.data.borrow_mut();
324         let mut destination_token_account =
325             StateWithExtensionsMut::<Account>::unpack(
 ↳ destination_token_account_data)?;
326
327         if destination_token_account.base.is_frozen() {
328             return Err(TokenError::AccountFrozen.into());
329         }
330
331         if destination_token_account.base.mint != *mint_info.key {
332             return Err(TokenError::MintMismatch.into());
333         }
334
335         let mut destination_confidential_transfer_account =
336             destination_token_account.get_extension_mut::<
 ↳ ConfidentialTransferAccount>()?;
337         destination_confidential_transfer_account.approved()?;
338
339         if !bool::from(&destination_confidential_transfer_account.
 ↳ allow_balance_credits) {
340             return Err(TokenError::
 ↳ ConfidentialTransferDepositsAndTransfersDisabled.into());
341         }
342
```

```
343         // Divide deposit into the low 16 and high 48 bits and
 ↳ then add to the appropriate pending
344         // ciphertexts
345         destination_confidential_transfer_account.
 ↳ pending_balance_lo = ops::add_to(
346             &destination_confidential_transfer_account.
 ↳ pending_balance_lo,
347             amount << PENDING_BALANCE_HI_BIT_LENGTH >>
 ↳ PENDING_BALANCE_HI_BIT_LENGTH,
348         )
349         .ok_or(ProgramError::InvalidInstructionData)?;
350
351         destination_confidential_transfer_account.
 ↳ pending_balance_hi = ops::add_to(
352             &destination_confidential_transfer_account.
 ↳ pending_balance_hi,
353             amount >> PENDING_BALANCE_LO_BIT_LENGTH,
354         )
355         .ok_or(ProgramError::InvalidInstructionData)?;
356
357         destination_confidential_transfer_account.
 ↳ pending_balance_credit_counter =
358             (u64::from(destination_confidential_transfer_account.
 ↳ pending_balance_credit_counter)
359                 .checked_add(1)
360                 .ok_or(ProgramError::InvalidInstructionData)?)
361             .into();
362
363         if u64::from(destination_confidential_transfer_account.
 ↳ pending_balance_credit_counter)
364             > u64::from(
365                 destination_confidential_transfer_account.
 ↳ maximum_pending_balance_credit_counter,
366             )
367         {
368             return Err(TokenError::
 ↳ MaximumPendingBalanceCreditCounterExceeded.into());
369         }
370     }
371
372     Ok(())
373 }
```

Listing 4: src/extension/confidential_transfer/processor.rs

```rust
fn process_withdraw(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    amount: u64,
    expected_decimals: u8,
    new_decryptable_available_balance: DecryptableBalance,
    proof_instruction_offset: i64,
) -> ProgramResult {
    let account_info_iter = &mut accounts.iter();
    let token_account_info = next_account_info(account_info_iter)?;
    let destination_token_account_info = next_account_info(account_info_iter)?;
    let mint_info = next_account_info(account_info_iter)?;
    let instructions_sysvar_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;
    let authority_info_data_len = authority_info.data_len();

    check_program_account(mint_info.owner)?;
    let mint_data = &mint_info.data.borrow_mut();
    let mint = StateWithExtensions::<Mint>::unpack(mint_data)?;

    if expected_decimals != mint.base.decimals {
        return Err(TokenError::MintDecimalsMismatch.into());
    }

    let previous_instruction =
        get_instruction_relative(proof_instruction_offset, instructions_sysvar_info)?;

    let proof_data = decode_proof_instruction::<WithdrawData>(
        ProofInstruction::VerifyWithdraw,
        &previous_instruction,
    )?;

    // Process source account
    {
        check_program_account(token_account_info.owner)?;
        let token_account_data = &mut token_account_info.data.borrow_mut();
        let mut token_account = StateWithExtensionsMut::<Account>::unpack(token_account_data)?;
```

```
414
415         Processor::validate_owner(
416             program_id,
417             &token_account.base.owner,
418             authority_info,
419             authority_info_data_len,
420             account_info_iter.as_slice(),
421         )?;
422
423         if token_account.base.is_frozen() {
424             return Err(TokenError::AccountFrozen.into());
425         }
426
427         if token_account.base.mint != *mint_info.key {
428             return Err(TokenError::MintMismatch.into());
429         }
430
431         let mut confidential_transfer_account =
432             token_account.get_extension_mut::<
 ↳ ConfidentialTransferAccount>()?;
433
434         confidential_transfer_account.available_balance =
435             ops::subtract_from(&confidential_transfer_account.
 ↳ available_balance, amount)
436                 .ok_or(ProgramError::InvalidInstructionData)?;
437
438         if confidential_transfer_account.available_balance !=
 ↳ proof_data.final_ciphertext {
439             return Err(TokenError::
 ↳ ConfidentialTransferBalanceMismatch.into());
440         }
441
442         confidential_transfer_account.
 ↳ decryptable_available_balance =
443             new_decryptable_available_balance;
444     }
445
446     //
447     // Finished with the source token account at this point. Drop
 ↳ all references to it to avoid a
448     // double borrow if the source and destination accounts are
 ↳ the same
449     //
450
```

```
451     // Process destination account
452     {
453         check_program_account(destination_token_account_info.owner
 ↳ )?;
454         let destination_token_account_data = &mut
 ↳ destination_token_account_info.data.borrow_mut();
455         let mut destination_token_account =
456             StateWithExtensionsMut::<Account>::unpack(
 ↳ destination_token_account_data)?;
457
458         if destination_token_account.base.is_frozen() {
459             return Err(TokenError::AccountFrozen.into());
460         }
461
462         if destination_token_account.base.mint != *mint_info.key {
463             return Err(TokenError::MintMismatch.into());
464         }
465
466         // Wrapped SOL withdrawals are not supported because
 ↳ lamports cannot be apparated.
467         assert!(!destination_token_account.base.is_native());
468         destination_token_account.base.amount =
 ↳ destination_token_account
469             .base
470             .amount
471             .checked_add(amount)
472             .ok_or(TokenError::Overflow)?;
473
474         destination_token_account.pack_base();
475     }
476
477     Ok(())
478 }
```

Listing 5: src/extension/confidential_transfer/processor.rs

```
482 fn process_transfer(
483     program_id: &Pubkey,
484     accounts: &[AccountInfo],
485     new_source_decryptable_available_balance: DecryptableBalance,
486     proof_instruction_offset: i64,
487 ) -> ProgramResult {
488     let account_info_iter = &mut accounts.iter();
489     let token_account_info = next_account_info(account_info_iter)
```

```
 ↳ ?;
490     let destination_token_account_info = next_account_info(
 ↳ account_info_iter)?;
491     let mint_info = next_account_info(account_info_iter)?;
492     let instructions_sysvar_info = next_account_info(
 ↳ account_info_iter)?;
493     let authority_info = next_account_info(account_info_iter)?;
494 
495     check_program_account(mint_info.owner)?;
496     let mint_data = &mint_info.data.borrow_mut();
497     let mint = StateWithExtensions::<Mint>::unpack(mint_data)?;
498     let confidential_transfer_mint = mint.get_extension::<
 ↳ ConfidentialTransferMint>()?;
499 
500     let previous_instruction =
501         get_instruction_relative(proof_instruction_offset,
 ↳ instructions_sysvar_info)?;
502 
503     if let Ok(transfer_fee_config) = mint.get_extension::<
 ↳ TransferFeeConfig>() {
504         // mint is extended for fees
505         let proof_data = decode_proof_instruction::<
 ↳ TransferWithFeeData>(
506             ProofInstruction::VerifyTransferWithFee,
507             &previous_instruction,
508         )?;
509 
510         if proof_data.transfer_with_fee_pubkeys.auditor_pubkey
511             != confidential_transfer_mint.
 ↳ auditor_encryption_pubkey
512         {
513             return Err(TokenError::
 ↳ ConfidentialTransferElGamalPubkeyMismatch.into());
514         }
515 
516         // `withdraw_withheld_authority` ElGamal pubkey in proof
 ↳ data and mint must match
517         if proof_data
518             .transfer_with_fee_pubkeys
519             .withdraw_withheld_authority_pubkey
520             != confidential_transfer_mint.
 ↳ withdraw_withheld_authority_encryption_pubkey
521         {
522             return Err(TokenError::
```

```
 ↳ ConfidentialTransferElGamalPubkeyMismatch.into());
523         }
524
525         // fee parameters in proof data and mint must match
526         let epoch = Clock::get()?.epoch;
527         let (maximum_fee, transfer_fee_basis_points) =
528             if u64::from(transfer_fee_config.newer_transfer_fee.
 ↳ epoch) < epoch {
529                 (
530                     u64::from(transfer_fee_config.
 ↳ older_transfer_fee.maximum_fee),
531                     u16::from(
532                         transfer_fee_config
533                             .older_transfer_fee
534                             .transfer_fee_basis_points,
535                     ),
536                 )
537             } else {
538                 (
539                     u64::from(transfer_fee_config.
 ↳ newer_transfer_fee.maximum_fee),
540                     u16::from(
541                         transfer_fee_config
542                             .newer_transfer_fee
543                             .transfer_fee_basis_points,
544                     ),
545                 )
546             };
547
548         if u64::from(proof_data.fee_parameters.maximum_fee) !=
 ↳ maximum_fee
549             || u16::from(proof_data.fee_parameters.
 ↳ fee_rate_basis_points)
550                 != transfer_fee_basis_points
551         {
552             return Err(TokenError::FeeParametersMismatch.into());
553         }
554
555         let source_ciphertext_lo = EncryptedBalance::from((
556             proof_data.ciphertext_lo.commitment,
557             proof_data.ciphertext_lo.source_handle,
558         ));
559         let source_ciphertext_hi = EncryptedBalance::from((
560             proof_data.ciphertext_hi.commitment,
```

```
561                 proof_data.ciphertext_hi.source_handle,
562             ));
563
564             process_source_for_transfer(
565                 program_id,
566                 token_account_info,
567                 mint_info,
568                 authority_info,
569                 account_info_iter.as_slice(),
570                 &proof_data.transfer_with_fee_pubkeys.source_pubkey,
571                 &source_ciphertext_lo,
572                 &source_ciphertext_hi,
573                 new_source_decryptable_available_balance,
574             )?;
575
576             let destination_ciphertext_lo = EncryptedBalance::from((
577                 proof_data.ciphertext_lo.commitment,
578                 proof_data.ciphertext_lo.destination_handle,
579             ));
580             let destination_ciphertext_hi = EncryptedBalance::from((
581                 proof_data.ciphertext_hi.commitment,
582                 proof_data.ciphertext_hi.destination_handle,
583             ));
584
585             let fee_ciphertext = if token_account_info.key ==
 ↳ destination_token_account_info.key {
586                 None
587             } else {
588                 Some(proof_data.fee_ciphertext)
589             };
590
591             process_destination_for_transfer(
592                 destination_token_account_info,
593                 mint_info,
594                 &proof_data.transfer_with_fee_pubkeys.
 ↳ destination_pubkey,
595                 &destination_ciphertext_lo,
596                 &destination_ciphertext_hi,
597                 fee_ciphertext,
598             )?;
599         } else {
600             // mint is not extended for fees
601             let proof_data = decode_proof_instruction::<TransferData>(
602                 ProofInstruction::VerifyTransfer,
```

```
603             &previous_instruction,
604         )?;
605 
606         if proof_data.transfer_pubkeys.auditor_pubkey
607             != confidential_transfer_mint.
 ↳ auditor_encryption_pubkey
608         {
609             return Err(TokenError::
 ↳ ConfidentialTransferElGamalPubkeyMismatch.into());
610         }
611 
612         let source_ciphertext_lo = EncryptedBalance::from((
613             proof_data.ciphertext_lo.commitment,
614             proof_data.ciphertext_lo.source_handle,
615         ));
616         let source_ciphertext_hi = EncryptedBalance::from((
617             proof_data.ciphertext_hi.commitment,
618             proof_data.ciphertext_hi.source_handle,
619         ));
620 
621         process_source_for_transfer(
622             program_id,
623             token_account_info,
624             mint_info,
625             authority_info,
626             account_info_iter.as_slice(),
627             &proof_data.transfer_pubkeys.source_pubkey,
628             &source_ciphertext_lo,
629             &source_ciphertext_hi,
630             new_source_decryptable_available_balance,
631         )?;
632 
633         let destination_ciphertext_lo = EncryptedBalance::from((
634             proof_data.ciphertext_lo.commitment,
635             proof_data.ciphertext_lo.destination_handle,
636         ));
637         let destination_ciphertext_hi = EncryptedBalance::from((
638             proof_data.ciphertext_hi.commitment,
639             proof_data.ciphertext_hi.destination_handle,
640         ));
641 
642         process_destination_for_transfer(
643             destination_token_account_info,
644             mint_info,
```

```
645             &proof_data.transfer_pubkeys.destination_pubkey,
646             &destination_ciphertext_lo,
647             &destination_ciphertext_hi,
648             None,
649         )?;
650     }
651
652     Ok(())
653 }
```

Listing 6: src/extension/confidential_transfer/processor.rs

```
657 fn process_destination_for_transfer(
658 ...
659    check_program_account(destination_token_account_info.owner)?;
660     let destination_token_account_data = &mut
 ↳ destination_token_account_info.data.borrow_mut();
661     let mut destination_token_account =
662         StateWithExtensionsMut::<Account>::unpack(
 ↳ destination_token_account_data)?;
663
664     if destination_token_account.base.is_frozen() {
665         return Err(TokenError::AccountFrozen.into());
666     }
667
668     if destination_token_account.base.mint != *mint_info.key {
669         return Err(TokenError::MintMismatch.into());
670     }
671
672     let mut destination_confidential_transfer_account =
673         destination_token_account.get_extension_mut::<
 ↳ ConfidentialTransferAccount>()?;
674     destination_confidential_transfer_account.approved()?;
675
676     if !bool::from(&destination_confidential_transfer_account.
 ↳ allow_balance_credits) {
677         return Err(TokenError::
 ↳ ConfidentialTransferDepositsAndTransfersDisabled.into());
678     }
679
680     if *destination_encryption_pubkey !=
 ↳ destination_confidential_transfer_account.encryption_pubkey
681     {
682         return Err(TokenError::
```

```
 ↳ ConfidentialTransferElGamalPubkeyMismatch.into());
683     }
```

Recommendation:

1. Ensure that the Confidential Transfer Extension transfer instruction checks if the tokens are non-transferable.
2. Apply the recommendation from HAL-01 to stop users from depositing/withdrawing tokens to a different token account.

Remediation Plan:

SOLVED The Solana Labs team resolved this issue in commits:
- 6a102589ef8ae268cc07666b9ea45739c55fe9f0
- 92a8e6b2d0499d8812b4e74c8f1f01a2a362dc4b
- b973e474683ede678ddf3789f80c7060ef136afc

Now the program prevents confidential transfers, withdraws and deposits of non-transferable tokens.

3.3 (HAL-03) MEMO REQUIREMENT CAN BE BYPASSED - LOW

Description:

The token-2022 program also introduces a memo_transfer extension, allowing developers to require users include a memo with their transfer. Similar to HAL-02 the Confidential Transfer Extension transfer instruction fails to validate if the destination token account requires a memo to be included in the transaction.

Code location:

Listing 7: token/program-2022/src/processor.rs

```
388         if memo_required(&destination_account) {
389             check_previous_sibling_instruction_is_memo()?;
390         }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Validate if the destination account requires a memo when doing a Confidential Transfer.

Remediation Plan:

SOLVED The Solana Labs team resolved this issue in commit 69c7fe465041661f98800c0ff3599d904e6494ff by checking if the destination account requires a memo for transfers.

3.4 (HAL-04) Amount Casting - Imprecision/Overflow - LOW

Description:

The amount_to_ui_amount, try_ui_amount_into_amount and ui_amount_to_amount instructions take an u64 amount and f64 amount, do some arithmetic operations and cast the result to f64 and u64 respectively. When converting/casting between types, an “overflow”/wrapping or mismatch may occur and result in logic bugs.

Code Location:

Listing 8: /extension/interest_bearing_mint/mod.rs (Lines 86-87)

```
80      pub fn amount_to_ui_amount(
81          &self,
82          amount: u64,
83          decimals: u8,
84          unix_timestamp: i64,
85      ) -> Option<String> {
86          let scaled_amount_with_interest =
87              (amount as f64) * self.total_scale(decimals,
↳ unix_timestamp)?;
88          Some(scaled_amount_with_interest.to_string())
89      }
```

Listing 9: /extension/interest_bearing_mint/mod.rs (Lines 102-109)

```
93      pub fn try_ui_amount_into_amount(
94          &self,
95          ui_amount: &str,
96          decimals: u8,
97          unix_timestamp: i64,
98      ) -> Result<u64, ProgramError> {
99          let scaled_amount = ui_amount
100             .parse::<f64>()
101             .map_err(|_| ProgramError::InvalidArgument)?;
102         let amount = scaled_amount
```

```
103             / self
104                 .total_scale(decimals, unix_timestamp)
105                 .ok_or(ProgramError::InvalidArgument)?;
106         if amount > (u64::MAX as f64) || amount < (u64::MIN as f64
 ↳ ) || amount.is_nan() {
107             Err(ProgramError::InvalidArgument)
108         } else {
109             Ok(amount.round() as u64) // this is important, if you
 ↳  round earlier, you'll get wrong "inf" answers
110         }
111     }
```

Listing 10: token/program-2022/src/lib.rs (Line 31)

```
30 pub fn ui_amount_to_amount(ui_amount: f64, decimals: u8) -> u64 {
31     (ui_amount * 10_usize.pow(decimals as u32) as f64) as u64
32 }
```

Listing 11: token/program-2022/src/lib.rs (Line 36)

```
35 pub fn amount_to_ui_amount(amount: u64, decimals: u8) -> f64 {
36     amount as f64 / 10_usize.pow(decimals as u32) as f64
37 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

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

Remediation Plan:

ACKNOWLEDGED: The Solana Labs Team acknowledged this finding.

3.5 (HAL-05) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL

Description:

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

Code Location:

- program-2022/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:

ACKNOWLEDGED: The Solana Labs Team acknowledged this finding.

3.6 (HAL-06) ZERO AMOUNT PROCESSING - INFORMATIONAL

Various functions in the Token-2022 program allow for zero amount processing. All these functions take the u64 amount parameter and use it for further
processing. None of the functions, however, do the sanity check of the parameter's value; therefore it is possible to process any arbitrary amount, including 0.

Instructions:

1. process_transfer
2. process_approve
3. process_mint_to
4. process_burn
5. process_deposit
6. process_withdraw

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Validate the amount parameter before processing the instruction further in order to avoid doing unnecessary calculations when amount is 0.

Remediation Plan:

PENDING: An issue has been opened, and the team is investigating.

3.7 (HAL-07) MINT INITIALIZATION FLOW - INFORMATIONAL

Description:

Developers should be aware that when creating a mint, the creation of the system account, initialization of mint extensions and mint initialization must be submitted in the same transaction. Failure to do so can allow attackers to initialize extensions with themselves as authorities and in the worst-case scenario take control of the mint.

Location:

- Token-2022 documentation

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Ensure Token-2022 documentation is updated and developers are educated on how to initialize mints correctly.

Remediation Plan:

SOLVED: The Solana Labs Team updated the Token-2022 documentation.

AUTOMATED TESTING

4.1 AUTOMATED VULNERABILITY SCANNING

Description:

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

Results:

The results below have been found to be false positives.

```
==========This account may be UNTRUSTFUL!==========
Found a potential vulnerability at line 101, column 21 in token/program-2022/src/extension/confidential_transfer/processor.rs
The account info is not trustful:

 95|        decryptable_zero_balance,
 96|        maximum_pending_balance_credit_counter,
 97|    }: &ConfigureAccountInstructionData,
 98|) -> ProgramResult {
 99|    let account_info_iter = &mut accounts.iter();
 100|    let token_account_info = next_account_info(account_info_iter)?;
>101|    let mint_info = next_account_info(account_info_iter)?;
 102|    let authority_info = next_account_info(account_info_iter)?;
 103|    let authority_info_data_len = authority_info.data_len();
 104|
 105|    check_program_account(token_account_info.owner)?;
 106|    let token_account_data = &mut token_account_info.data.borrow_mut();
 107|    let mut token_account = StateWithExtensionsMut::<Account>::unpack(token_account_data)?;
>>>Stack Trace:
>>>spl_token_2022::processor::Processor::process::ha3071b0a558bfea8 [token/program-2022/src/entrypoint.rs:17]
>>>  spl_token_2022::extension::confidential_transfer::processor::process_instruction::h002d8c63c87c3dce [token/program-2022/src/processor.rs:1280]
>>>    spl_token_2022::extension::confidential_transfer::processor::process_configure_account::h74af7c8200389147 [token/program-2022/src/extension/confidential_transfer/processor.rs:1178]
```

```
==========This account may be UNTRUSTFUL!==========
Found a potential vulnerability at line 952, column 25 in token/program-2022/src/processor.rs
The account info is not trustful:

 946|        program_id: &Pubkey,
 947|        accounts: &[AccountInfo],
 948|        freeze: bool,
 949|    ) -> ProgramResult {
 950|        let account_info_iter = &mut accounts.iter();
 951|        let source_account_info = next_account_info(account_info_iter)?;
>952|        let mint_info = next_account_info(account_info_iter)?;
 953|        let authority_info = next_account_info(account_info_iter)?;
 954|        let authority_info_data_len = authority_info.data_len();
 955|
 956|        let mut source_account_data = source_account_info.data.borrow_mut();
 957|        let mut source_account =
 958|            StateWithExtensionsMut::<Account>::unpack(&mut source_account_data)?;
>>>Stack Trace:
>>>spl_token_2022::processor::Processor::process::ha3071b0a558bfea8 [token/program-2022/src/entrypoint.rs:17]
>>>  spl_token_2022::processor::Processor::process_toggle_freeze_account::h87c5dcfbf81ebf6f [token/program-2022/src/processor.rs:1246]
```

```
============This account may be UNTRUSTFUL!================
Found a potential vulnerability at line 118, column 25 in token/program-2022/src/processor.rs
The account info is not trustful:

 112|        accounts: &[AccountInfo],
 113|        owner: Option<&Pubkey>,
 114|        rent_sysvar_account: bool,
 115|    ) -> ProgramResult {
 116|        let account_info_iter = &mut accounts.iter();
 117|        let new_account_info = next_account_info(account_info_iter)?;
>118|        let mint_info = next_account_info(account_info_iter)?;
 119|        let owner = if let Some(owner) = owner {
 120|            owner
 121|        } else {
 122|            next_account_info(account_info_iter)?.key
 123|        };
 124|        let new_account_info_data_len = new_account_info.data_len();
>>>Stack Trace:
>>>spl_token_2022::processor::Processor::process::ha3071b0a558bfea8 [token/program-2022/src/entrypoint.rs:17]
>>>  spl_token_2022::processor::Processor::process_initialize_account3::h38e31420a40623aa [token/program-2022/src/processor.rs:1198]
>>>    spl_token_2022::processor::Processor::_process_initialize_account::hd0a3275bd4237de2 [token/program-2022/src/processor.rs:199]
```

```
============This account may be UNTRUSTFUL!================
Found a potential vulnerability at line 459, column 22 in token/program-2022/src/processor.rs
The account info is not trustful:

 453|            StateWithExtensionsMut::<Account>::unpack(&mut source_account_data)?;
 454|
 455|        if source_account.base.is_frozen() {
 456|            return Err(TokenError::AccountFrozen.into());
 457|        }
 458|
>459|        if let Some((mint_info, expected_decimals)) = expected_mint_info {
 460|            if !cmp_pubkeys(&source_account.base.mint, mint_info.key) {
 461|                return Err(TokenError::MintMismatch.into());
 462|            }
 463|
 464|            let mint_data = mint_info.data.borrow();
 465|            let mint = StateWithExtensions::<Mint>::unpack(&mint_data)?;
>>>Stack Trace:
>>>spl_token_2022::processor::Processor::process::ha3071b0a558bfea8 [token/program-2022/src/entrypoint.rs:17]
>>>  spl_token_2022::processor::Processor::process_approve::hba7bece2443e15ef [token/program-2022/src/processor.rs:1254]
```

```
============This account may be UNTRUSTFUL!================
Found a potential vulnerability at line 1071, column 25 in token/program-2022/src/processor.rs
The account info is not trustful:

 1065|            .map(|_| ())
 1066|    }
 1067|
 1068|    /// Processes an [AmountToUiAmount](enum.TokenInstruction.html) instruction
 1069|    pub fn process_amount_to_ui_amount(accounts: &[AccountInfo], amount: u64) -> ProgramResult {
 1070|        let account_info_iter = &mut accounts.iter();
>1071|        let mint_info = next_account_info(account_info_iter)?;
 1072|        check_program_account(mint_info.owner)?;
 1073|
 1074|        let mint_data = mint_info.data.borrow();
 1075|        let mint = StateWithExtensions::<Mint>::unpack(&mint_data)
 1076|            .map_err(|_| Into::<ProgramError>::into(TokenError::InvalidMint))?;
 1077|        let ui_amount = if let Ok(extension) = mint.get_extension::<InterestBearingConfig>() {
>>>Stack Trace:
>>>spl_token_2022::processor::Processor::process::ha3071b0a558bfea8 [token/program-2022/src/entrypoint.rs:17]
>>>  spl_token_2022::processor::Processor::process_amount_to_ui_amount::hf5a732a706973b0b [token/program-2022/src/processor.rs:1299]
```

```
============This account may be UNTRUSTFUL!================
Found a potential vulnerability at line 1093, column 25 in token/program-2022/src/processor.rs
The account info is not trustful:

 1087|        Ok(())
 1088|    }
 1089|
 1090|    /// Processes an [AmountToUiAmount](enum.TokenInstruction.html) instruction
 1091|    pub fn process_ui_amount_to_amount(accounts: &[AccountInfo], ui_amount: &str) -> ProgramResult {
 1092|        let account_info_iter = &mut accounts.iter();
>1093|        let mint_info = next_account_info(account_info_iter)?;
 1094|        check_program_account(mint_info.owner)?;
 1095|
 1096|        let mint_data = mint_info.data.borrow();
 1097|        let mint = StateWithExtensions::<Mint>::unpack(&mint_data)
 1098|            .map_err(|_| Into::<ProgramError>::into(TokenError::InvalidMint))?;
 1099|        let amount = if let Ok(extension) = mint.get_extension::<InterestBearingConfig>() {
>>>Stack Trace:
>>>spl_token_2022::processor::Processor::process::ha3071b0a558bfea8 [token/program-2022/src/entrypoint.rs:17]
>>>  spl_token_2022::processor::Processor::process_ui_amount_to_amount::h86b3d115567ba0a0 [token/program-2022/src/processor.rs:1303]
```

4.2 AUTOMATED ANALYSIS

Description:

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

ID	package	Short Description
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2022-0046	rocksdb	Out-of-bounds read
RUSTSEC-2021-0139	ansi_term	Unmaintained
RUSTSEC-2020-0016	net2	Unmaintained
RUSTSEC-2020-0056	stdweb	Unmaintained

4.3 UNSAFE RUST CODE DETECTION

Description:

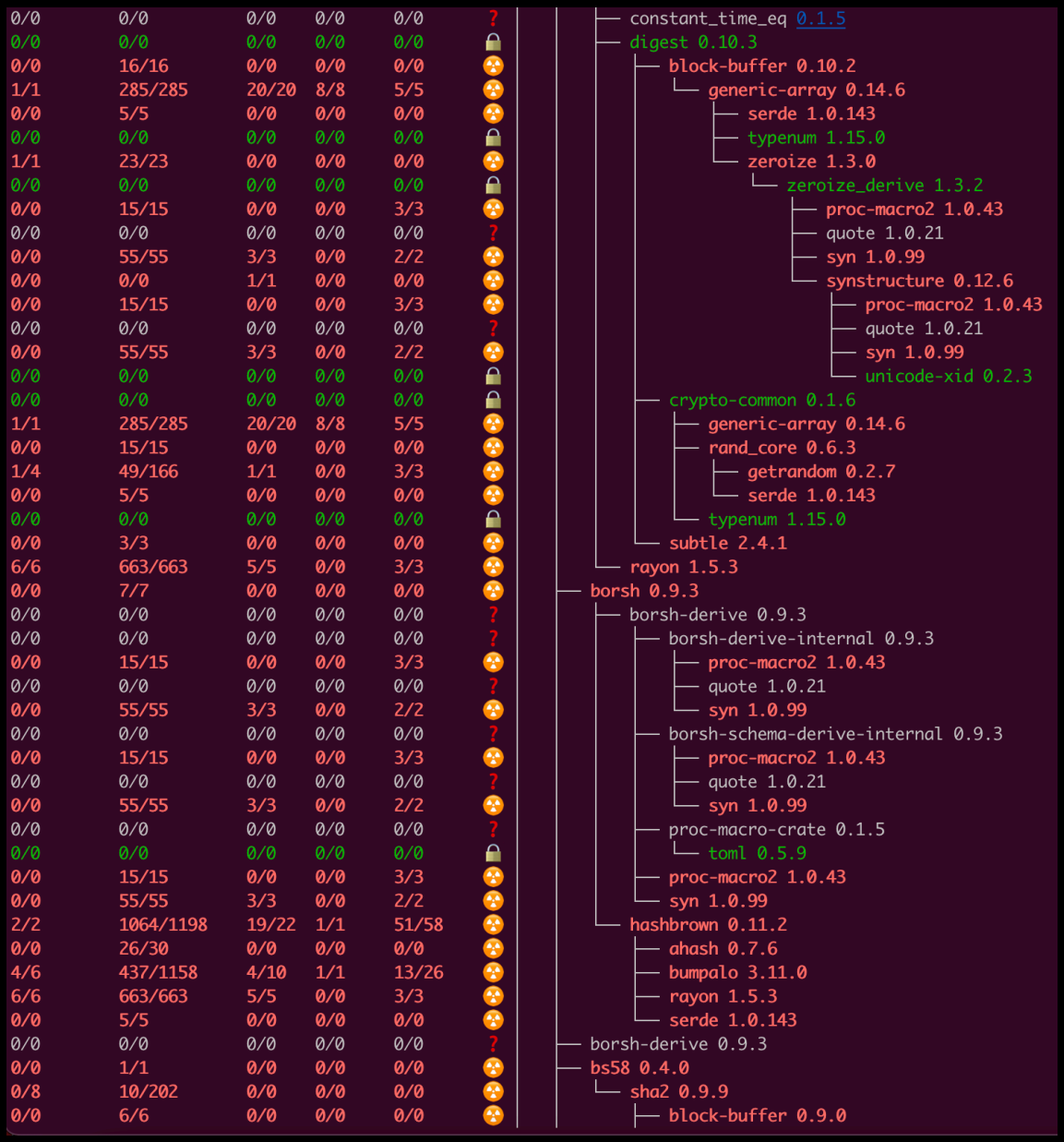
Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

AUTOMATED TESTING

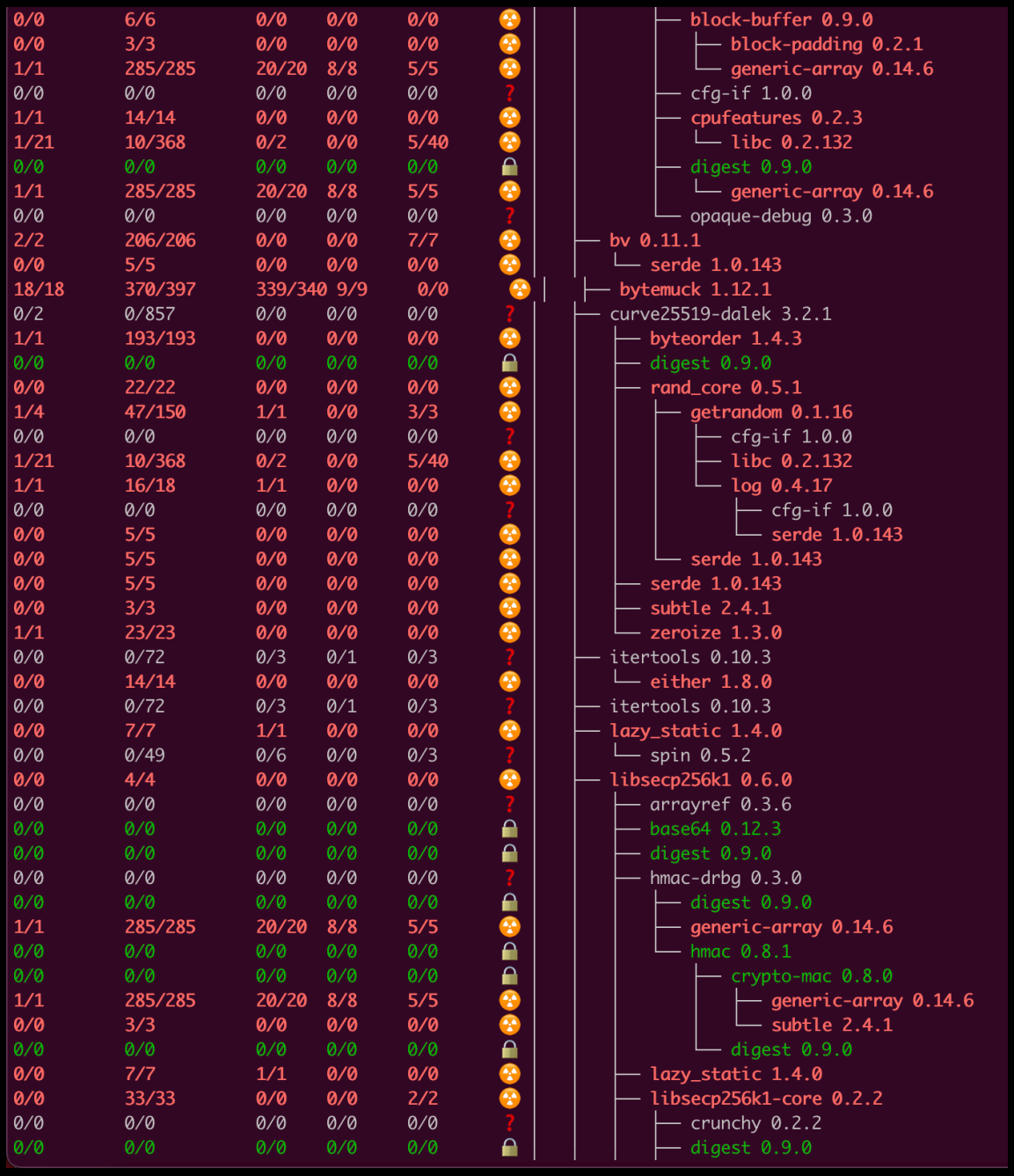
Results:

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

Functions  Expressions  Impls  Traits  Methods  Dependency

0/18       0/468        0/56   0/0     0/0      ❓ spl-token-2022 0.4.0
0/0        0/0          0/0    0/0     0/0      ❓ ├── arrayref 0.3.6
18/18      370/397      339/340 9/9     0/0      ☢ ├── bytemuck 1.12.1
0/0        0/0          0/0    0/0     0/0      ❓ │   └── bytemuck_derive 1.2.1
0/0        15/15        0/0    0/0     3/3      ☢ │       ├── proc-macro2 1.0.43
0/0        4/4          0/0    0/0     0/0      ☢ │       │   └── unicode-ident 1.0.3
0/0        0/0          0/0    0/0     0/0      ❓ │       ├── quote 1.0.21
0/0        15/15        0/0    0/0     3/3      ☢ │       │   └── proc-macro2 1.0.43
0/0        55/55        3/3    0/0     2/2      ☢ │       └── syn 1.0.99
0/0        15/15        0/0    0/0     3/3      ☢ │           ├── proc-macro2 1.0.43
0/0        0/0          0/0    0/0     0/0      ❓ │           ├── quote 1.0.21
0/0        4/4          0/0    0/0     0/0      ☢ │           └── unicode-ident 1.0.3
0/0        0/0          0/0    0/0     0/0      ❓ ├── num-derive 0.3.3
0/0        15/15        0/0    0/0     3/3      ☢ │   ├── proc-macro2 1.0.43
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── quote 1.0.21
0/0        55/55        3/3    0/0     2/2      ☢ │   └── syn 1.0.99
0/0        6/12         0/0    0/0     0/0      ☢ ├── num-traits 0.2.15
0/0        0/0          0/0    0/0     0/0      ❓ ├── num_enum 0.5.7
0/0        0/0          0/0    0/0     0/0      ❓ │   └── num_enum_derive 0.5.7
0/0        0/0          0/0    0/0     0/0      ❓ │       ├── proc-macro-crate 1.2.1
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
1/21       10/368       0/2    0/0     5/40     ☢ │       │   │       ├── libc 0.2.132
0/1        0/399        0/7    0/1     0/13     ❓ │       │   │       └── smallvec 1.9.0
0/0        5/5          0/0    0/0     0/0      ☢ │       │   │           └── serde 1.0.143
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │               └── serde_derive 1.0.143
0/0        15/15        0/0    0/0     3/3      ☢ │       │   │                   ├── proc-macro2 1.0.43
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │                   ├── quote 1.0.21
0/0        55/55        3/3    0/0     2/2      ☢ │       │   │                   └── syn 1.0.99
0/0        0/0          0/0    0/0     0/0      ❓ │       │   ├── thiserror 1.0.32
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │   └── thiserror-impl 1.0.32
0/0        15/15        0/0    0/0     3/3      ☢ │       │   │       ├── proc-macro2 1.0.43
0/0        0/0          0/0    0/0     0/0      ❓ │       │   │       ├── quote 1.0.21
0/0        55/55        3/3    0/0     2/2      ☢ │       │   │       └── syn 1.0.99
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 1.9.1
0/1        0/1367       0/24   0/1     0/69     ❓ │       │       │   ├── hashbrown 0.12.3
0/0        26/30        0/0    0/0     0/0      ☢ │       │       │   │   ├── ahash 0.7.6
0/0        26/30        0/0    0/0     0/0      ☢ │       │       │   │   ├── ahash 0.7.6
1/4        49/166       1/1    0/0     3/3      ☢ │       │       │   │   │   ├── getrandom 0.2.7
0/0        0/0          0/0    0/0     0/0      ❓ │       │       │   │   │   │   ├── cfg-if 1.0.0
1/21       10/368       0/2    0/0     5/40     ☢ │       │       │   │   │   │   └── libc 0.2.132
1/1        76/118       4/6    0/0     2/3      ☢ │       │       │   │   │   ├── once_cell 1.13.1
0/0        5/5          0/0    0/0     0/0      ☢ │       │       │   │   │   └── serde 1.0.143
4/6        437/1158     4/10   1/1     13/26    ☢ │       │       │   │   ├── bumpalo 3.11.0
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 1.0.143
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
```

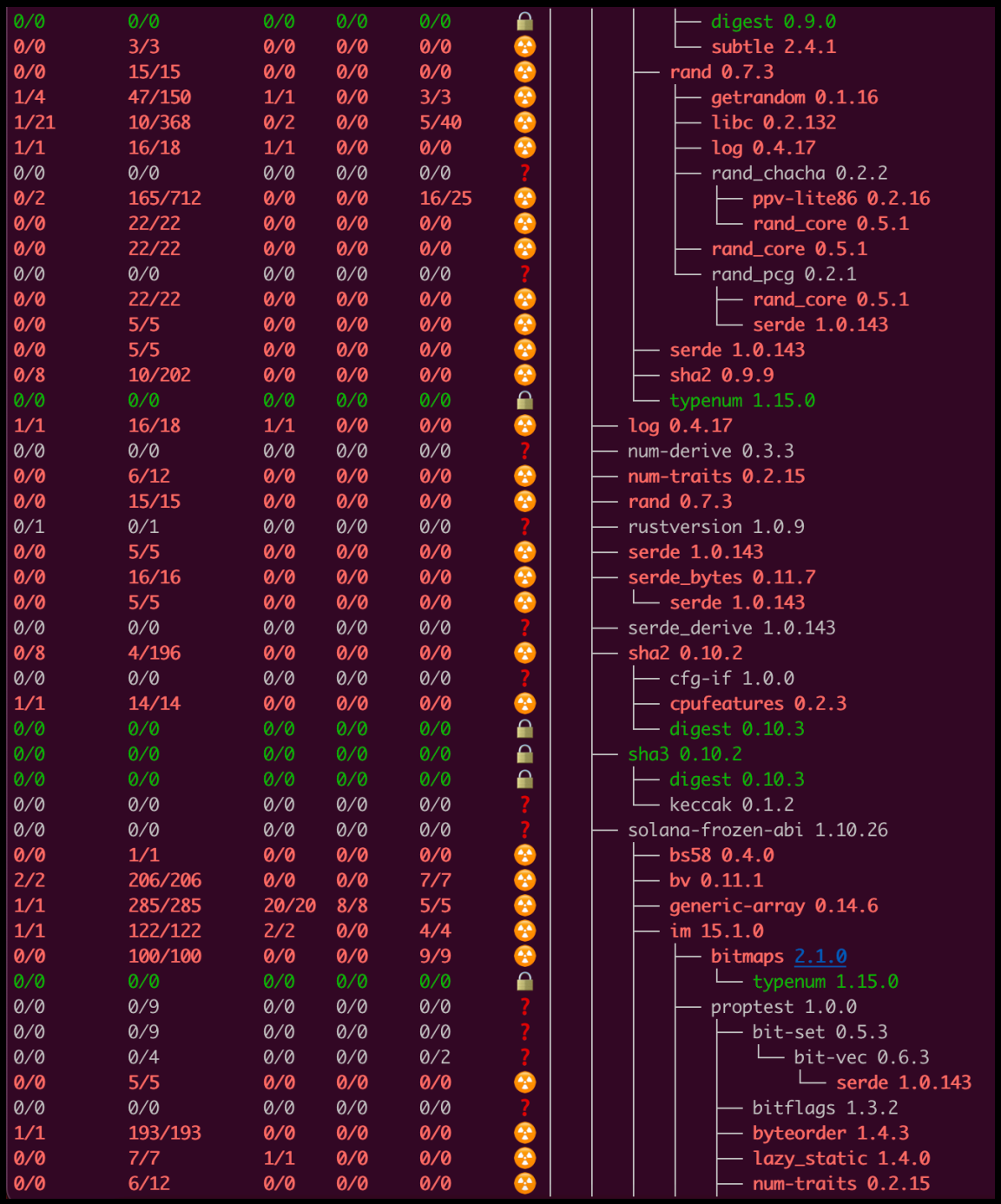
AUTOMATED TESTING



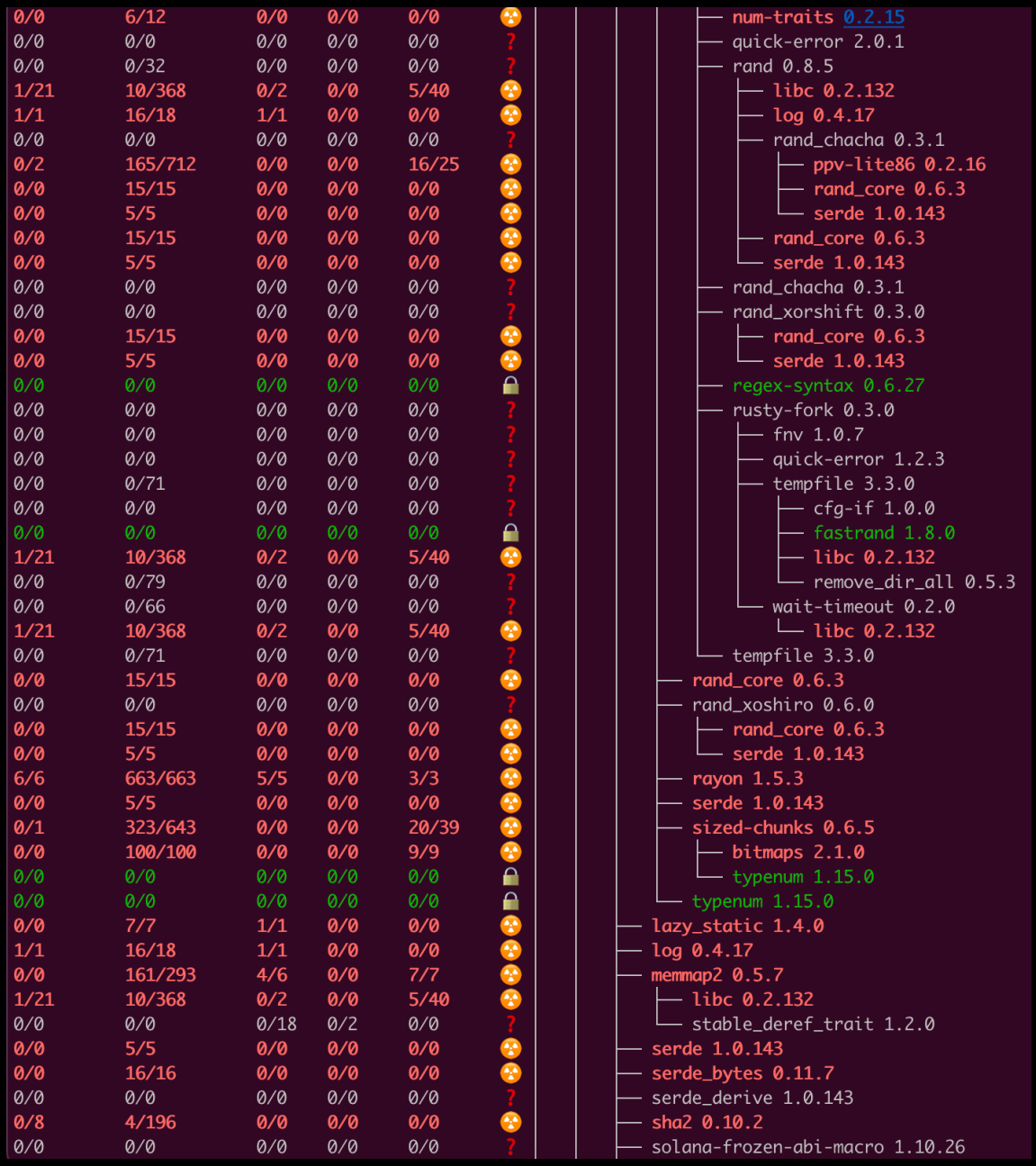
AUTOMATED TESTING



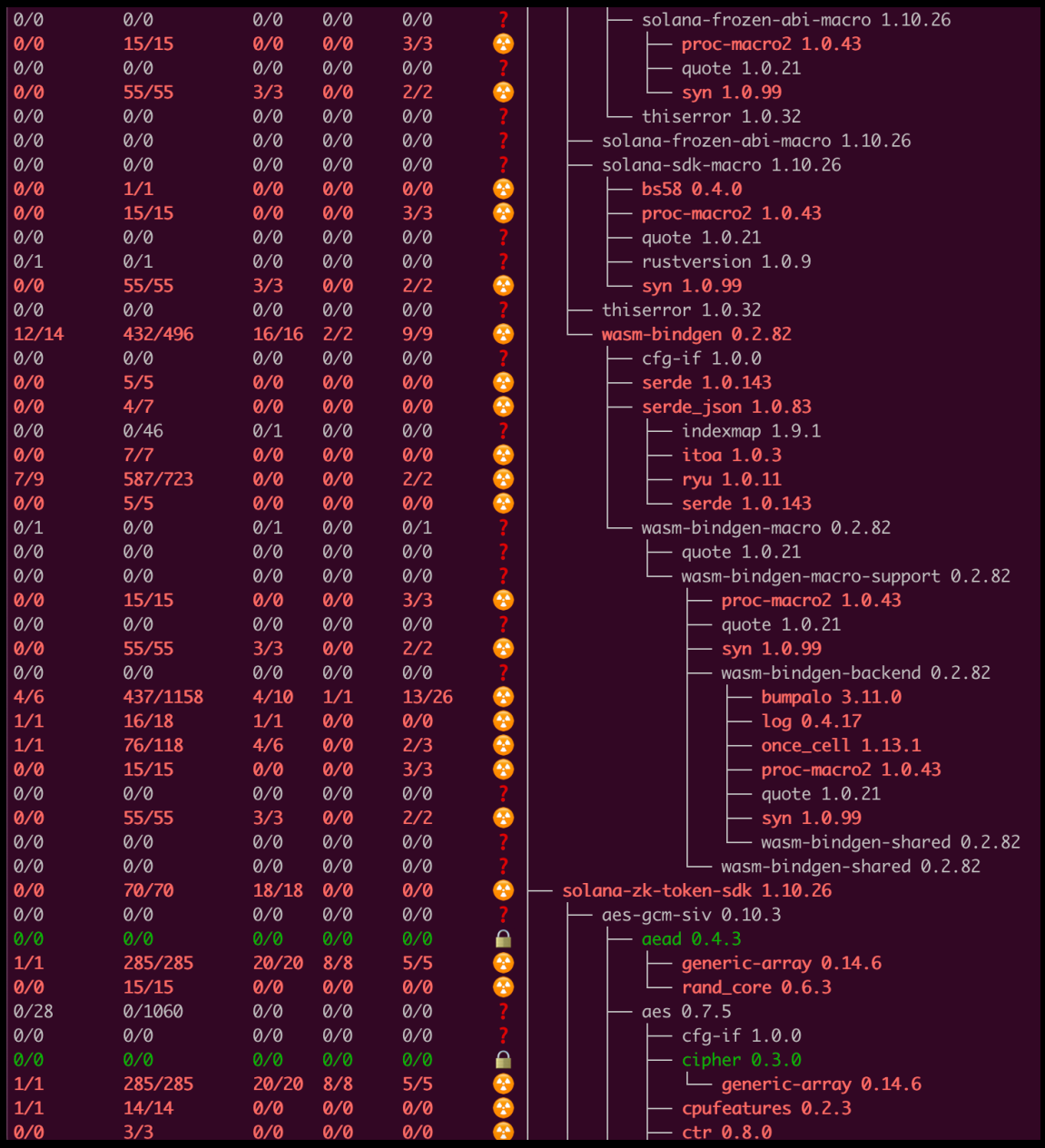
AUTOMATED TESTING



AUTOMATED TESTING



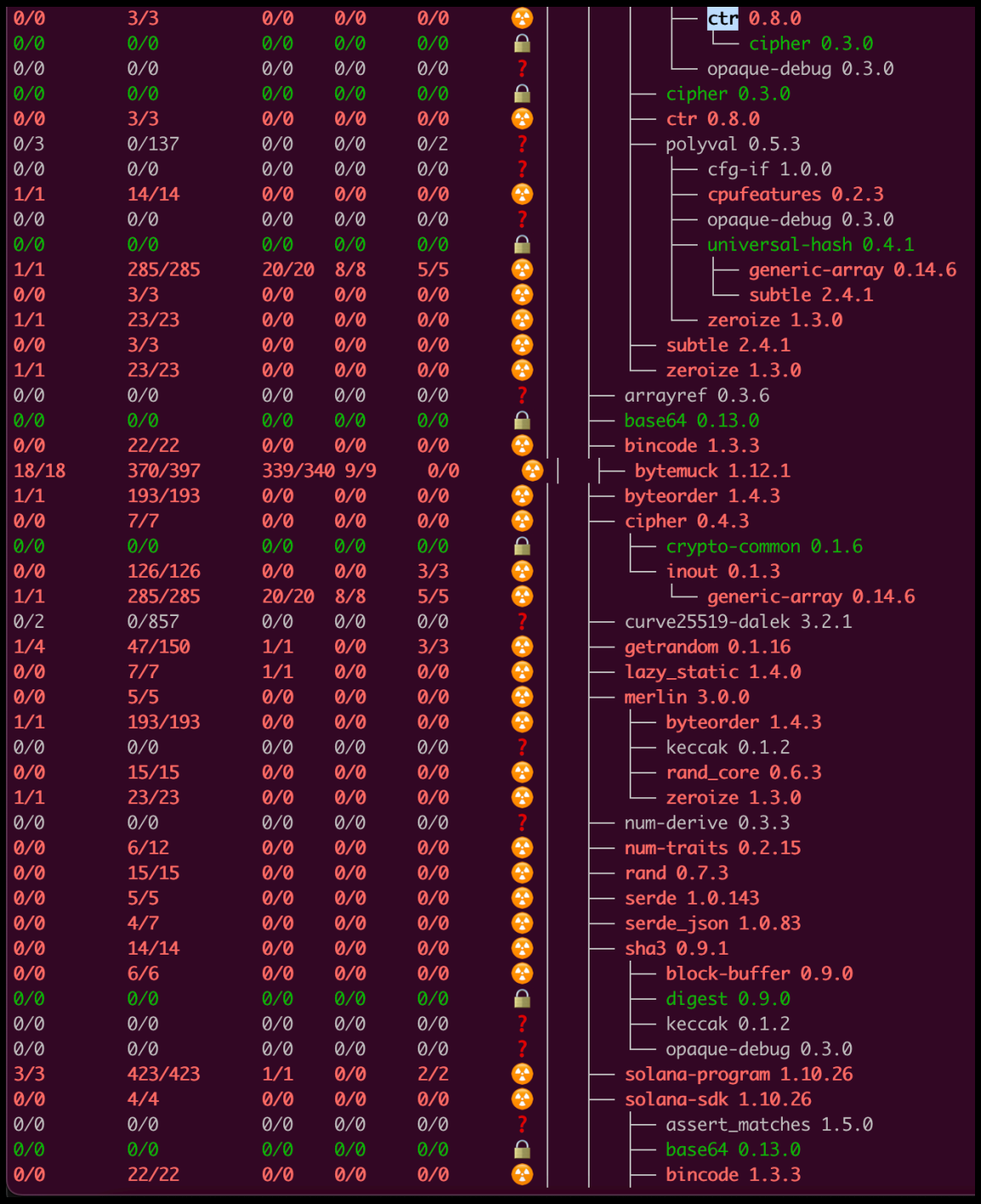
AUTOMATED TESTING



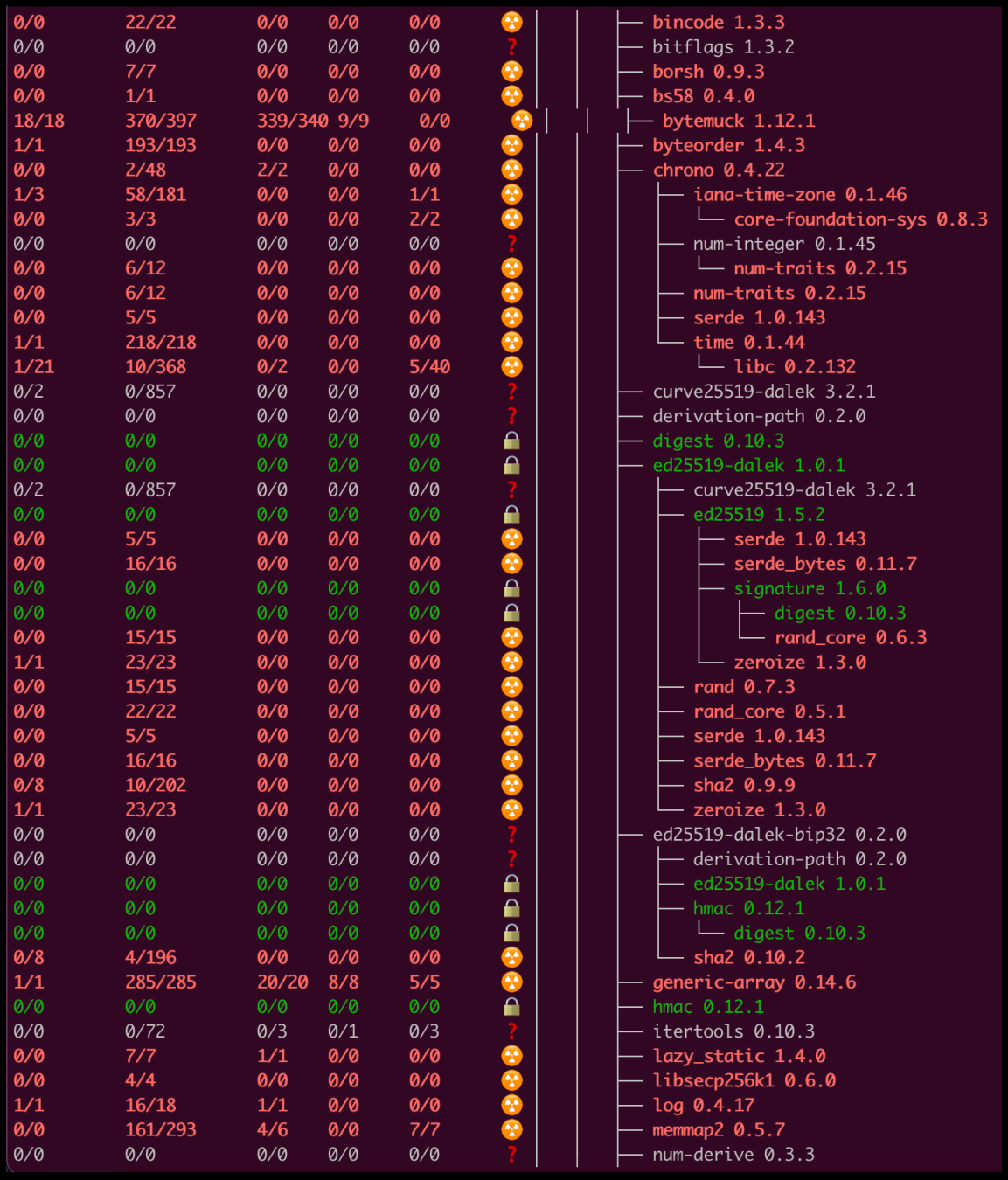
0/0	0/0	0/0	0/0	0/0	?
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	55/55	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	1/1	0/0	0/0	0/0	☢
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/1	0/1	0/0	0/0	0/0	?
0/0	55/55	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
12/14	432/496	16/16	2/2	9/9	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	5/5	0/0	0/0	0/0	☢
0/0	4/7	0/0	0/0	0/0	☢
0/0	0/46	0/1	0/0	0/0	?
0/0	7/7	0/0	0/0	0/0	☢
7/9	587/723	0/0	0/0	2/2	☢
0/0	5/5	0/0	0/0	0/0	☢
0/1	0/0	0/1	0/0	0/1	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	55/55	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
4/6	437/1158	4/10	1/1	13/26	☢
1/1	16/18	1/1	0/0	0/0	☢
1/1	76/118	4/6	0/0	2/3	☢
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	55/55	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	70/70	18/18	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	🔒
1/1	285/285	20/20	8/8	5/5	☢
0/0	15/15	0/0	0/0	0/0	☢
0/28	0/1060	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	🔒
1/1	285/285	20/20	8/8	5/5	☢
1/1	14/14	0/0	0/0	0/0	☢
0/0	3/3	0/0	0/0	0/0	☢

```
    ├── solana-frozen-abi-macro 1.10.26
    │   ├── proc-macro2 1.0.43
    │   ├── quote 1.0.21
    │   └── syn 1.0.99
    └── thiserror 1.0.32
├── solana-frozen-abi-macro 1.10.26
├── solana-sdk-macro 1.10.26
    ├── bs58 0.4.0
    ├── proc-macro2 1.0.43
    ├── quote 1.0.21
    ├── rustversion 1.0.9
    └── syn 1.0.99
├── thiserror 1.0.32
└── wasm-bindgen 0.2.82
    ├── cfg-if 1.0.0
    ├── serde 1.0.143
    ├── serde_json 1.0.83
    │   ├── indexmap 1.9.1
    │   ├── itoa 1.0.3
    │   ├── ryu 1.0.11
    │   └── serde 1.0.143
    └── wasm-bindgen-macro 0.2.82
        ├── quote 1.0.21
        └── wasm-bindgen-macro-support 0.2.82
            ├── proc-macro2 1.0.43
            ├── quote 1.0.21
            ├── syn 1.0.99
            ├── wasm-bindgen-backend 0.2.82
            │   ├── bumpalo 3.11.0
            │   ├── log 0.4.17
            │   ├── once_cell 1.13.1
            │   ├── proc-macro2 1.0.43
            │   ├── quote 1.0.21
            │   ├── syn 1.0.99
            │   └── wasm-bindgen-shared 0.2.82
            └── wasm-bindgen-shared 0.2.82
├── solana-zk-token-sdk 1.10.26
    ├── aes-gcm-siv 0.10.3
    │   ├── aead 0.4.3
    │   │   ├── generic-array 0.14.6
    │   │   └── rand_core 0.6.3
    │   ├── aes 0.7.5
    │   │   ├── cfg-if 1.0.0
    │   │   ├── cipher 0.3.0
    │   │   │   └── generic-array 0.14.6
    │   │   ├── cpufeatures 0.2.3
    │   │   ├── ctr 0.8.0
```

AUTOMATED TESTING



AUTOMATED TESTING



THANK YOU FOR CHOOSING

// HALBORN