// HALBORN

Solana Labs - Runtime 77a56b0 -> 124aaa9

L1 Security Assessment

Prepared by: **Halborn**

Date of Engagement: **June 12th, 2023 - July 14th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/12/2023	Isabel Burruezo
0.2	Document Updates	07/11/2023	Isabel Burruezo
0.3	Draft Version	07/12/2023	Isabel Burruezo
0.4	Draft Review	07/12/2023	Piotr Cielas
0.5	Draft Review	07/12/2023	Gabi Urrutia
1.0	Remediation Plan	08/22/2023	Isabel Burruezo
1.1	Remediation Plan	08/30/2023	Isabel Burruezo
1.2	Remediation Plan Review	08/30/2023	Piotr Cielas
1.3	Remediation Plan Review	09/01/2023	Piotr Cielas

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 is an open-source project implementing a new, high-performance, permissionless blockchain. Changes in scope affected several modules, the most important ones are briefly described. Sealevel, Solana's parallel smart contracts runtime, is a concurrent transaction processor. Transactions specify their data dependencies upfront, and dynamic memory allocation is explicit. By separating program code from the state it operates on, the runtime can choreograph concurrent access.

Halborn conducted a security assessment on a set of changes to the Solana repository made between two different commits, beginning on June 12th, 2023 and ending on July 14th, 2023 . The security assessment was scoped to the updates to the master branch of the solana GitHub repository. Commit hashes and further details can be found in the **Scope** section of this report.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided five weeks for the engagement and assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing and smart-contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn did not identify any significant issues; however, some recommendations were given to reduce the likelihood and impact of risks, which were acknowledged by the Solana Labs team.

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 assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage 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 assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual 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)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Solana L1

- Repository: solana
  - start: 77a56b0
  - final: 124aaa9
- Modules in scope:
  1. program-runtime (solana/program-runtime/src)
  2. runtime (solana/runtime/src)
  3. bpf_loader (solana/programs/bpf_loader/src)

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

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING CARGO OVERFLOW CHECKS	Informational (0.0)	NOT APPLICABLE
(HAL-02) OPEN TO-DO	Informational (0.0)	ACKNOWLEDGED
(HAL-03) INCOMPLETE FUNCTIONALITY IMPLEMENTATION	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) MISSING CARGO OVERFLOW CHECKS - INFORMATIONAL (0.0)

Description:

We have noticed that the Cargo.toml file does not include the **overflow-checks=true** setting. By default, overflow checks are disabled in optimized release builds. Consequently, if an overflow occurs in release builds, it will be suppressed, resulting in unexpected application behavior. It is advisable to include the **overflow-checks=true** check in the Cargo.toml file, even if checked arithmetic is employed using checked_ or saturating_* functions.

Code Location:

- program-runtime/Cargo.toml
- programs/bpf_loader/Cargo.toml

BVSS:

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

Recommendation:

It is strongly advised to include the **overflow-checks=true** configuration under the release profile within your Cargo.toml file.

Remediation Plan:

NOT APPLICABLE: The code in scope for this audit does not use unchecked integer arithmetic, so Solana Labs has opted not to incur potential performance penalties by enabling overflow checks.

4.2 (HAL-02) OPEN TO-DO - INFORMATIONAL (0.0)

Description:

The transaction priority detail code is responsible for determining the priority and execution details of transactions in the network. The `get_transaction_priority_details` function is responsible for calculating and providing the priority details of a transaction. This information helps nodes and validators determine how to process the transaction.

An open to-do item was discovered in the `transaction_priority_details.rs` file.

The `round_compute_unit_price_enabled` parameter was introduced to the `get_transaction_priority_details()` and `process_compute_budget_instruction()` functions. However, currently, it is not utilized. Instead, a **TODO** comment in the call to the `process_instructions()` function indicates that it should be provided in the future.

Additionally, several other to-do items have been identified in the following files:

- runtime/src/serde_snapshot/newer.rs
- runtime/src/bank.rs

While none of these to-dos are considered security risks, it is crucial to implement all the required functionalities before undergoing an assessment. This measure ensures that unverified bugs do not arise in future implementations.

Code Location:

Listing 1: runtime/src/transaction_priority_details.rs (Lines 19,24,34)

```
16 pub trait GetTransactionPriorityDetails {
17     fn get_transaction_priority_details(
18         &self,
19         round_compute_unit_price_enabled: bool,
20     ) -> Option<TransactionPriorityDetails>;
21
22     fn process_compute_budget_instruction<'a>(
23         instructions: impl Iterator<Item = (&'a Pubkey, &'a
↳ CompiledInstruction)>,
24         _round_compute_unit_price_enabled: bool,
25     ) -> Option<TransactionPriorityDetails> {
26         let mut compute_budget = ComputeBudget::default();
27         let prioritization_fee_details = compute_budget
28             .process_instructions(
29                 instructions,
30                 true,  // use default units per instruction
31                 false, // stop supporting prioritization by
↳ request_units_deprecated instruction
32                 true,  // enable request heap frame instruction
33                 true,  // enable support set accounts data size
↳ instruction
34                 // TODO: round_compute_unit_price_enabled:
↳ bool
35             )
36             .ok()?;
37         Some(TransactionPriorityDetails {
38             priority: prioritization_fee_details.get_priority(),
39             compute_unit_limit: compute_budget.compute_unit_limit,
40         })
```

Listing 2: runtime/src/serde_snapshot/newer.rs (Line 218)

```
205 let epoch_reward_status = serializable_bank
206         .bank
207         .get_epoch_reward_status_to_serialize();
208     match get_serialize_bank_fields(
209         SerializableVersionedBank::from(fields),
210         SerializableAccountsDb::<'a, Self> {
211             accounts_db: &serializable_bank.bank.rc.accounts.
```

```
 ↳ accounts_db,
212             slot: serializable_bank.bank.rc.slot,
213             account_storage_entries: serializable_bank.
 ↳ snapshot_storages,
214             phantom: std::marker::PhantomData::default(),
215         },
216         // Additional fields, we manually store the lamps per
 ↳ signature here so that
217         // we can grab it on restart.
218         // TODO: if we do a snapshot version bump, consider moving
 ↳  this out.
219         lamports_per_signature,
220         None::<BankIncrementalSnapshotPersistence>,
221         serializable_bank
222             .bank
223             .get_epoch_accounts_hash_to_serialize()
224             .map(|epoch_accounts_hash| *epoch_accounts_hash.as_ref
 ↳ ()),
225         epoch_reward_status,
226     ) {
227         BankFieldsToSerialize::WithoutEpochRewardStatus(data) =>
 ↳ data.serialize(serializer),
228         BankFieldsToSerialize::WithEpochRewardStatus(data) => data
 ↳ .serialize(serializer),
229     }
```

Listing 3: runtime/src/bank.rs (Line 778)

```
777     is_delta,
778     // TODO: Confirm if all these fields are intentionally
 ↳ ignored!
779     builtin_programs: _,
780     runtime_config: _,
```

Listing 4: runtime/src/bank.rs (Lines 5583,5584)

```
5582 if !accounts_to_store.is_empty() {
5583     // TODO: Maybe do not call `store_accounts()` here.  Instead
  ↳ return `accounts_to_store`
5584     // and have `collect_rent_in_partition()` perform all the
  ↳ stores.
5585     let (_, measure) = measure!(self.store_accounts((
5586         self.slot(),
```

```
5587             &accounts_to_store[..],
5588             self.include_slot_in_hash()
5589         )));
5590         time_storing_accounts_us += measure.as_us();
5591     }
```

BVSS:

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

Recommendation:

Ensure that the pending to-do items are either implemented or evaluated for removal if they will not be incorporated into future releases.

Remediation Plan:

ACKNOWLEDGED: The Solana Labs team acknowledged this issue.

4.3 (HAL-03) INCOMPLETE FUNCTIONALITY IMPLEMENTATION - INFORMATIONAL (0.0)

Description:

In the accounts' module of Solana, the RewardInterval enum was introduced. In includes a variant called OutsideInterval, which represents the slot in the epoch that falls outside the reward distribution interval. This new enum value is used as a parameter in calls to functions such as load_accounts_with_fee_and_rent(), load_accounts(), and load_transaction_accounts().

It is important to note that the current implementation of this addition does not introduce any changes to the runtime, as it is still a work in progress. The RewardInterval enum is a preparatory step for future updates, indicating that the implementation is underway but not yet fully integrated or functional.

This still ongoing implementation also happens in the tiered accounts storage module. Its purpose can be found explained in the proposal tiered-accounts-db-storage; however, its implementation is still in progress and until it is not finalized it is not possible to verify its full impact as well as certain features such as the immutability of the account file once it is created.

It is important to mention that this can also be found in part of the **partitioned epoch rewards** implementation with the following functions in the bank and metrics:

- report_partitioned_reward_metrics
- partitioned_epoch_rewards_config
- partitioned_rewards_stake_account_stores_per_block
- get_reward_calculation_num_blocks
- set_epoch_reward_status_active

Code Location:

Listing 5: runtime/src/accounts.rs

```
85 pub(crate) enum RewardInterval {
86     /// the slot within the epoch is OUTSIDE the reward
 ↳ distribution interval
87     OutsideInterval,
88 }
```

Listing 6: runtime/src/accounts.rs (Line 1556)

```
1523 fn load_accounts_with_fee_and_rent(
1524     tx: Transaction,
1525     ka: &[TransactionAccount],
1526     lamports_per_signature: u64,
1527     rent_collector: &RentCollector,
1528     error_counters: &mut TransactionErrorMetrics,
1529     feature_set: &FeatureSet,
1530     fee_structure: &FeeStructure,
1531 ) -> Vec<TransactionLoadResult> {
1532     let mut hash_queue = BlockhashQueue::new(100);
1533     hash_queue.register_hash(&tx.message().recent_blockhash,
 ↳ lamports_per_signature);
1534     let accounts = Accounts::new_with_config_for_tests(
1535         Vec::new(),
1536         &ClusterType::Development,
1537         AccountSecondaryIndexes::default(),
1538         AccountShrinkThreshold::default(),
1539     );
1540     for ka in ka.iter() {
1541         accounts.store_for_tests(0, &ka.0, &ka.1);
1542     }
1543
1544     let ancestors = vec![(0, 0)].into_iter().collect();
1545     let sanitized_tx = SanitizedTransaction::
 ↳ from_transaction_for_tests(tx);
1546     accounts.load_accounts(
1547         &ancestors,
1548         &[sanitized_tx],
1549         vec![(Ok(()), None)],
1550         &hash_queue,
1551         error_counters,
1552         rent_collector,
```

```
553         feature_set,
554         fee_structure,
555         None,
556         RewardInterval::OutsideInterval,
557         &HashMap::new(),
558         &LoadedProgramsForTxBatch::default(),
```

Listing 7: runtime/src/accounts.rs (Lines 688,726)

```
677 pub(crate) fn load_accounts(
678     &self,
679     ancestors: &Ancestors,
680     txs: &[SanitizedTransaction],
681     lock_results: Vec<TransactionCheckResult>,
682     hash_queue: &BlockhashQueue,
683     error_counters: &mut TransactionErrorMetrics,
684     rent_collector: &RentCollector,
685     feature_set: &FeatureSet,
686     fee_structure: &FeeStructure,
687     account_overrides: Option<&AccountOverrides>,
688     in_reward_interval: RewardInterval,
689     program_accounts: &HashMap<Pubkey, &Pubkey>,
690     loaded_programs: &LoadedProgramsForTxBatch,
691 ) -> Vec<TransactionLoadResult> {
692     txs.iter()
693         .zip(lock_results)
694         .map(|etx| match etx {
695             (tx, (Ok(()), nonce)) => {
696                 let lamports_per_signature = nonce
697                     .as_ref()
698                     .map(|nonce| nonce.lamports_per_signature())
699                     .unwrap_or_else(|| {
700                         hash_queue.get_lamports_per_signature(tx.
 ↳ message().recent_blockhash())
701                     });
702                 let fee = if let Some(lamports_per_signature) =
 ↳ lamports_per_signature {
703                     Bank::calculate_fee(
704                         tx.message(),
705                         lamports_per_signature,
706                         fee_structure,
707                         feature_set.is_active(&
 ↳ use_default_units_in_fee_calculation::id()),
708                         !feature_set.is_active(&
```

```
 ↳ remove_deprecated_request_unit_ix::id()),
709                         feature_set.is_active(&
 ↳ remove_congestion_multiplier_from_fee_calculation::id()),
710                         feature_set.is_active(&
 ↳ enable_request_heap_frame_ix::id()) || self.accounts_db.
 ↳ expected_cluster_type() != ClusterType::MainnetBeta,
711                         feature_set.is_active(&
 ↳ add_set_tx_loaded_accounts_data_size_instruction::id()),
712                         feature_set.is_active(&
 ↳ include_loaded_accounts_data_size_in_fee_calculation::id()),
713                     )
714                 } else {
715                     return (Err(TransactionError::
 ↳ BlockhashNotFound), None);
716                 };
717
718                 let loaded_transaction = match self.
 ↳ load_transaction_accounts(
719                     ancestors,
720                     tx,
721                     fee,
722                     error_counters,
723                     rent_collector,
724                     feature_set,
725                     account_overrides,
726                     in_reward_interval,
```

Listing 8: runtime/src/accounts.rs (Lines 686,726)

```
677 fn load_transaction_accounts(
678     &self,
679     ancestors: &Ancestors,
680     tx: &SanitizedTransaction,
681     fee: u64,
682     error_counters: &mut TransactionErrorMetrics,
683     rent_collector: &RentCollector,
684     feature_set: &FeatureSet,
685     account_overrides: Option<&AccountOverrides>,
686     _reward_interval: RewardInterval,
687     program_accounts: &HashMap<Pubkey, &Pubkey>,
688     loaded_programs: &LoadedProgramsForTxBatch,
689  ...
```

Listing 9: runtime/src/tiered_storage/footer.rs

```
228 pub fn new_from_mmap(map: &Mmap) -> TsResult<&TieredStorageFooter>
 ↳  {
229     let offset = map.len().saturating_sub(FOOTER_TAIL_SIZE);
230     let (footer_size, offset) = get_type::<u64>(map, offset)?;
231     let (_footer_version, offset) = get_type::<u64>(map, offset)?;
232     let (magic_number, _offset) = get_type::<
 ↳ TieredStorageMagicNumber>(map, offset)?;
233
234     if *magic_number != TieredStorageMagicNumber::default() {
235         return Err(TieredStorageError::MagicNumberMismatch(
236             TieredStorageMagicNumber::default().0,
237             magic_number.0,
238         ));
239     }
240
241     let (footer, _offset) =
242         get_type::<TieredStorageFooter>(map, map.len().
 ↳ saturating_sub(*footer_size as usize))?;
243
244     Ok(footer)
245 }
```

BVSS:

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

Recommendation:

To ensure the robustness of a system, it is of utmost importance to implement all necessary functionalities prior to undergoing an assessment. This proactive approach helps to mitigate the possibility of encountering unverified bugs or issues in future implementations. It is recommended to complete all required functionalities, the system can be thoroughly evaluated for security, reliability, and overall effectiveness.

Remediation Plan:

ACKNOWLEDGED: The Solana Labs team acknowledged this issue.

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 code for.

5.1 IMPLICIT HANDLING OF DELAY VISIBILITY TOMBSTONE AND USABLE ENTRIES

Description:

In commit b20024c7
the is_implicit_delay_visibility_tombstone() function was introduced to address the issue of duplicating code for handling delay visibility tombstones in the transaction batch cache. This addition enables the code to be refactored and consolidated, resulting in updated call sites.

In addition, in commit a649459f is_entry_usable was introduced to enhance the extract() function. This newly added function serves as a mechanism to filter and select suitable program entries. It evaluates the usability of a program entry by considering factors such as expiration status, matching criteria, and whether it is unloaded **LoadedProgramType**. By utilizing is_entry_usable, the extract() function can efficiently filter and choose appropriate program entries that meet the specified criteria. This improvement ensures the extraction process is optimized and selects only the desired program entries.

Thorough review and testing were conducted on these changes to guarantee that the newly added features produce the desired and expected outcomes.

Results:

No vulnerabilities were identified.

MANUAL TESTING

```
[+] Testing is implicit..
--->Tombstone Closed<---
effective_slot: 3
deployment slot: 3
implicit delay  in slot 3? : false
--->Tombstone FailedVerification<---
effective_slot: 2
deployment slot: 2
implicit delay  in slot 3? : false
--->Builtin<---
effective_slot: 4
deployment slot: 3
implicit delay  in slot 3? : false
--->Unloaded<---
effective_slot: 4
deployment slot: 3
implicit delay  in slot 3? : true
--->TestLoaded<---
effective_slot: 4
deployment slot: 3
is implicit delay in slot 3? : true
test loaded_programs::tests::test_is_implicit ... ok
```

MANUAL TESTING

```
[*] Testing usable entries for Builting Porgram
current slot: 0
Match criteria: NoCriteria
entry not expiration
result entry usable?: true
current slot: 9
Match criteria: NoCriteria
entry not expiration
result entry usable?: true
current slot: 13
Match criteria: NoCriteria
entry not expiration
result entry usable?: true
current slot: 0
Match criteria: Tombstone
entry not expiration
result entry usable?: false
current slot: 9
Match criteria: Tombstone
entry not expiration
result entry usable?: false
current slot: 0
Match criteria: DeployedOnOrAfterSlot(0)
entry not expiration
result entry usable?: true
current slot: 4
Match criteria: DeployedOnOrAfterSlot(12)
entry not expiration
result entry usable?: false
current slot: 13
Match criteria: DeployedOnOrAfterSlot(5)
entry not expiration
result entry usable?: false
current slot: 13
Match criteria: DeployedOnOrAfterSlot(3)
entry not expiration
result entry usable?: true
```

MANUAL TESTING

```
[*]Testing usable entries for Unloaded Program
current slot: 0
Match criteria: NoCriteria
entry not expiration
current slot: 1
Match criteria: NoCriteria
entry not expiration
current slot: 1
Match criteria: Tombstone
entry not expiration
current slot: 1
Match criteria: DeployedOnOrAfterSlot(0)
entry not expiration
[*]Testing usable entries for Tombstone Closed
current slot: 0
Match criteria: NoCriteria
entry not expiration
current slot: 1
Match criteria: Tombstone
entry not expiration
current slot: 1
Match criteria: NoCriteria
entry not expiration
current slot: 1
Match criteria: DeployedOnOrAfterSlot(0)
entry not expiration
current slot: 1
Match criteria: DeployedOnOrAfterSlot(1)
entry not expiration
```

5.2 PRUNE ON FEATURE SET TRANSITION

Description:

In commit e55a582e the latest updates involve two key modifications. Firstly, the LoadedPrograms::prune_feature_set_transition() function was added. Secondly, the Bank::apply_builtin_program_feature_transitions() now includes a call to create_program_runtime_environment().

These changes were implemented to ensure that the cache remains up-to-date by removing any obsolete entries following the feature transition. However, it is worth noting that the updates do not include recompiling the entries before reaching the epoch limit.

Thorough testing was conducted on these changes to ensure the cache really remains current by eliminating any obsolete entries after a feature transition. This testing was crucial to prevent inconsistencies and vulnerabilities that could arise from outdated code not aligning with the updated feature set, thereby mitigating the risks of unexpected behavior and security loopholes.

Results:

No vulnerabilities were identified.

```
prune feature set transition
Entry program : LoadedProgramType::Unloaded
retain: false
Entry program : LoadedProgramType::FailedVerification
retain: false
Entry program : LoadedProgramType::TestLoaded
retain: false
Prunes of loaded programs cache after prune: 3
cache empty?: true
```

MANUAL TESTING

```
[+] Prune feature set transition...
Entry program : LoadedProgramType::Closed
retain: true
Entry program : LoadedProgramType::TestLoaded
retain: false
Entry program : LoadedProgramType::DelayVisibility
retain: true
Entry program : LoadedProgramType::Builtin
retain: true
Prunes of loaded programs cache after prune: 1
```

5.3 PURGE INCOMPLETE BANK SNAPSHOTS

Description:

In commit 4dddc840 the purge_incomplete_bank_snapshots() function was introduced, which serves the purpose of deleting all incomplete bank snapshots during startup. This deletion process occurs only once for both the validator and the general ledger tool. The addition of this function addresses the issue of retaining unnecessary bank snapshots. Once a snapshot is archived, the older snapshots become obsolete and no longer serve any purpose.

Furthermore, bank snapshots include hard links to account archive files to facilitate faster startup. However, if the accounts are stored on a RAM disk, these snapshots can artificially occupy space. Hence, it is crucial to promptly free up this space. The newly added function ensures the efficient cleanup of incomplete bank snapshots, enabling resources to be utilized optimally.

Several tests have been performed to ensure that the bank snapshots that can be purged are indeed incomplete and adequate to avoid inconsistencies as well as possible future vulnerabilities.

Results:

No vulnerabilities were identified.

```
[+] Testing purge incomplete bank snapshot for complete snapshots
 getting bank snapshot directory for slot 1 ..
 getting bank snapshot directory for slot 2 ..
 getting bank snapshot directory for slot 3 ..
 getting bank snapshot directory for slot 4 ..
[+] Purge incomplete bank snapshots
Bank snapshot directory read
bank snapshot status is complete?: true
bank snapshot status is complete?: true
bank snapshot status is complete?: true
bank snapshot status is complete?: true
bank snapshot directory for slot 1 exists after purging?: true
bank snapshot directory for slot 2 exists after purging?: true
bank snapshot directory for slot 3 exists after purging?: true
bank snapshot directory for slot 4 exists after purging?: true
test snapshot_utils::tests::test_purge_incomplete_bank_snapshots_expectFail_with_complete_snapshots ... ok
```

5.4 LOADED PROGRAMS CACHE AND TRANSACTION BATCH CACHE

Description:

In commit 8313409c changes were introduced in order to replace the usage of the executor cache with the LoadedPrograms cache and update the transaction batch cache with the transaction results, among others. These modifications aim to improve the efficiency and reliability of the system by leveraging the LoadedPrograms cache and addressing various issues.

The mentioned code changes have undergone a thorough review and testing process to ensure they are robust and minimize the potential for vulnerabilities or security risks.

Results:

No vulnerabilities were identified.

```
[*]Replenish
Programs Loaded for Tx Batch replenished
Default Programs Modified by Tx
Default Programs Updated only for global cache
[*] Process message
[*] New Invoke Conext
Program id: 1111111QLbz7JHiBTspS962RLKV8GndWFwiEaqKM
Program not precompile
Number of instruction accounts: 3
[*]Pushing the instruction account
[*]Pushing the instruction account
[*]Pushing the instruction account
[*]Process instruction
[*]Process executable chain
[*] Find in Programs Loaded for Tx Batch
Implicit delay visibility tombstone: false
Program entry effective and cloned!
Entry found from Porgrams Loaded for Tx Batch: LoadedProgramType::Closed
```

AUTOMATED TESTING

6.1 AUTOMATED ANALYSIS

Description:

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

Results:

ID	package	Short Description
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2023-0001	tokio	reject_remote_clients Configuration corruption

6.2 UNSAFE RUST CODE DETECTION

Description:

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

Results:

No unsafe code blocks were identified in the packages in scope and their dependencies.

THANK YOU FOR CHOOSING

// HALBORN