Solana Runtime

Audit

Presented by:

OtterSec contact@osec.io

Harrison Green hgarrereyn@osec.io

Alec Petridis alec@osec.io

Durable Nonce Review

June 6th - June 10th, 2022

Table of Contents

01 | Scope & Timeline

Overview

OtterSec performed an assessment of several PRs related to the recent durable nonce bug and loss of consensus.

The audit scope consists of the following areas:

1. **Solana Core PRs**
    a. https://github.com/solana-labs/solana/pull/25744
    b. https://github.com/solana-labs/solana/pull/25788
    c. https://github.com/solana-labs/solana/pull/25789
    d. https://github.com/solana-labs/solana/pull/25831
    e. https://github.com/solana-labs/solana/pull/25832
2. **Durable Nonce Code in Solana Core** (non exhaustive list)
    a. sdk/program/src/system_instruction.rs
    b. sdk/src/nonce_account.rs
    c. runtime/src/bank.rs
    d. runtime/src/nonce_keyed_account.rs
    e. runtime/src/system_instruction_processor.rs

We focused specifically on runtime-critical code such as transaction processing. Off-chain systems, e.g. CLI compatibility and documentation is out of scope.

This assessment began on **June 6th, 2022** and ended on **June 10th, 2022**.

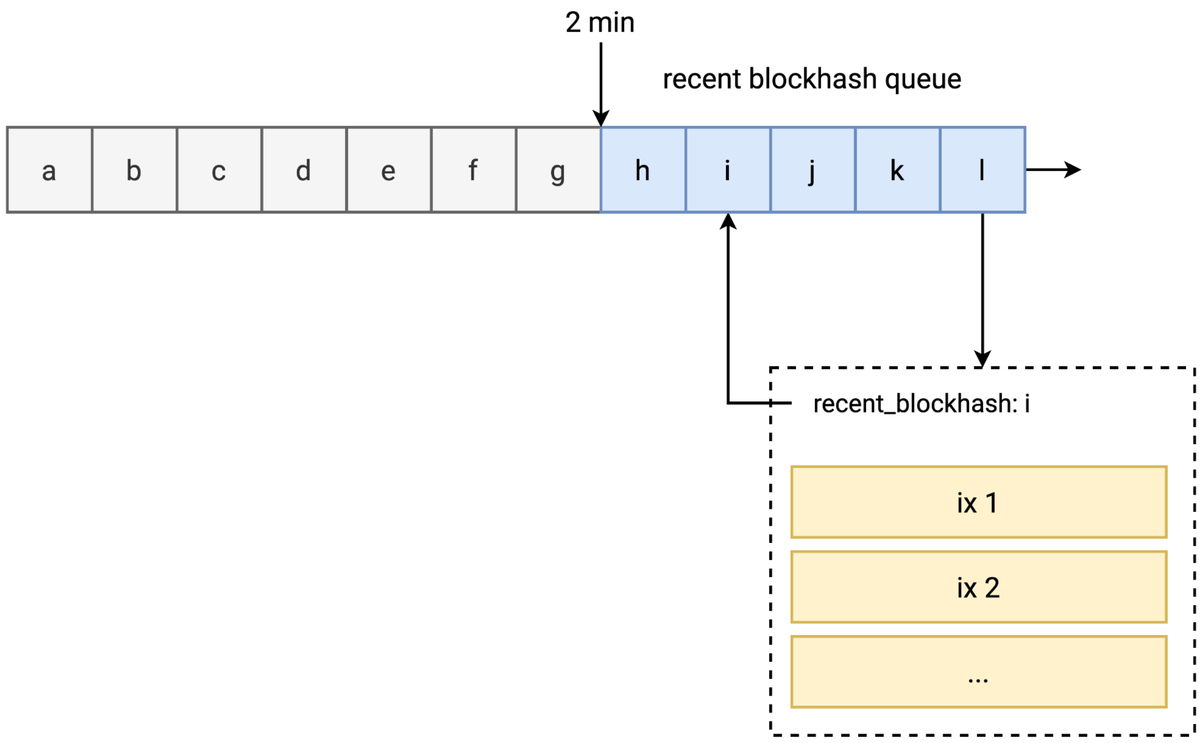
Critical vulnerabilities were communicated to the team before the delivery of the report to speed up remediation.

02 | Root Cause Analysis

Normal Transactions

To prevent transaction replay on the Solana blockchain, validator nodes need to be able to detect if a transaction has already been included. Requiring validator nodes to store the entire history of transactions since genesis would be computationally expensive and therefore transactions store a `recent_blockhash` field which is used to truncate the historical context that validator nodes need to store.

In a normal transaction, the `recent_blockhash` field points to a blockhash no more than 2 minutes older than the current top blockhash:

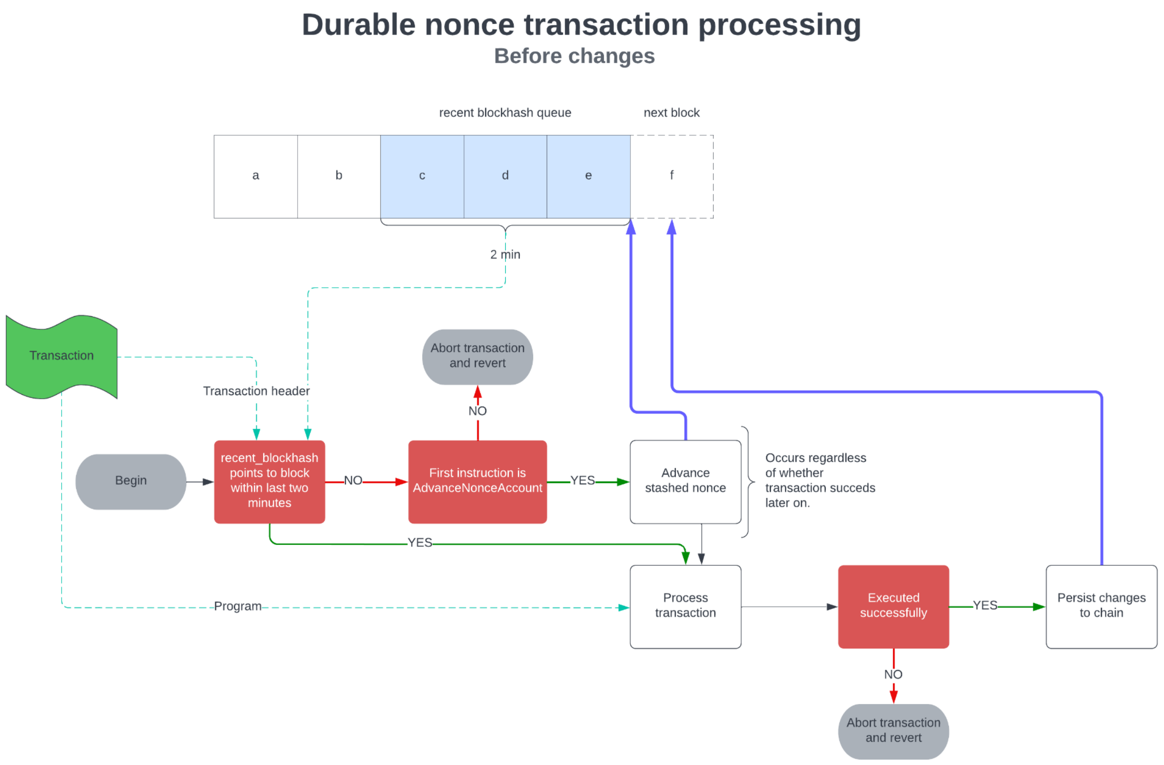


Transactions with a recent_blockhash field older than 2 minutes are “expired” and will be ignored by validators. With this mechanism, validator nodes need only to search the most recent 2 minutes of transaction history for duplicate transactions. Since the `recent_blockhash` field is included for transaction signature computation, an adversary cannot easily modify the recent_blockhash field to force double-execution (without the ability to forge the account signatures).

NonceAccount Overview

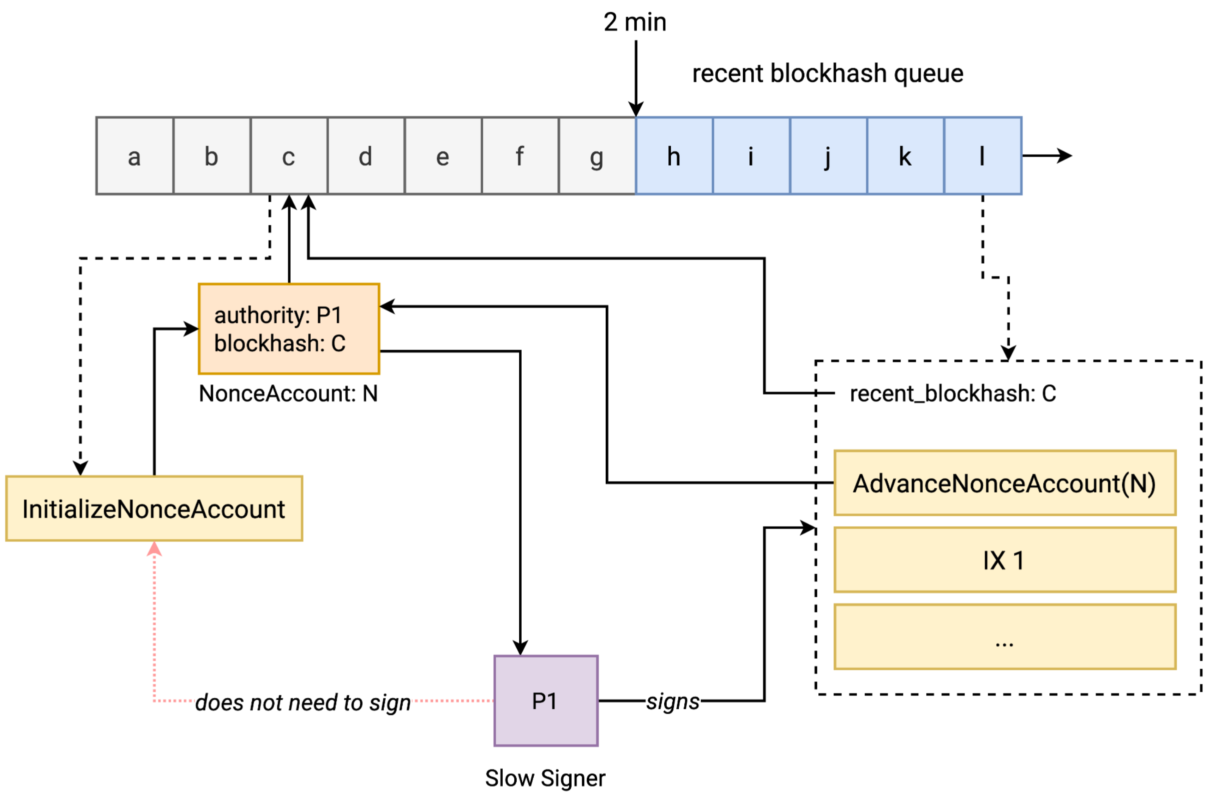
One problem that arises from this scenario is that transaction signature computation must take less than 2 minutes, otherwise, the `recent_blockhash` will be expired by the time validator nodes see the transaction. This is not always feasible when signature computation uses offline methods.

To alleviate this problem, Solana provides a `NonceAccount` which can be used to sign transactions using a nonce stashed in the past. When a transaction intends to use a `NonceAccount`, it must include the `AdvanceNonceAccount` instruction at the start of the transaction, which will advance the account's stashed nonce to the last blockhash and indicate to the Bank runtime code that this transaction can execute as a nonce transaction.


Durable nonce transaction processing
Before changes
recent blockhash queue
next block
a
b
c
d
e
f
2 min
Transaction
Transaction header
Abort transaction and revert
NO
Begin
recent_blockhash points to block within last two minutes
NO
First instruction is AdvanceNonceAccount
YES
Advance stashed nonce
Occurs regardless of whether transaction succeds later on.
YES
Program
Process transaction
Executed successfully
YES
Persist changes to chain
NO
Abort transaction and revert

Note that these transaction errors occur at program runtime, once the transaction has been already included in the block.

The intended usage of a `NonceAccount` is shown below:



In slot C, a user creates a new account owned by the system program and invokes `InitializeNonceAccount`, providing the authority for whom this `NonceAccount` can be used.

Importantly, the authority (in this diagram, P1) does not need to sign for the initial instruction, rather any other account (e.g. an online signer) can do so.

The `NonceAccount` acts as a *one time proof of single-execution*. At any point in the future, the authority can issue a transaction that refers to this `NonceAccount` instead of a `recent_blockhash`, however in doing so, it will irrevocably "consume" the `NonceAccount` so that this `NonceAccount` becomes invalid for the original transaction.

In theory, there are many different ways to implement "referring to a NonceAccount" and "consuming a NonceAccount." The fundamental principle is that *a nonce transaction should only be valid the first time it uses a NonceAccount and no subsequent times.*

For example, one approach would be to generate a unique value in the `NonceAccount` and refer to both the `NonceAccount` and the unique value in the transaction. During nonce transaction processing, the values would be compared (failing if they don't match) and the `NonceAccount` would obtain a new unique value, thereby invalidating the original transaction from executing a second time (because the values no longer match).

In Solana, the `NonceAccount` uses block hashes as a generator for unique values. Specifically, during initialization, the `NonceAccount` stores the most recent blockhash. When a transaction is constructed that uses this NonceAccount, it stores that same blockhash in the `recent_blockhash` field and includes a special `AdvanceNonceInstruction` as the first instruction which points to the `NonceAccount`.

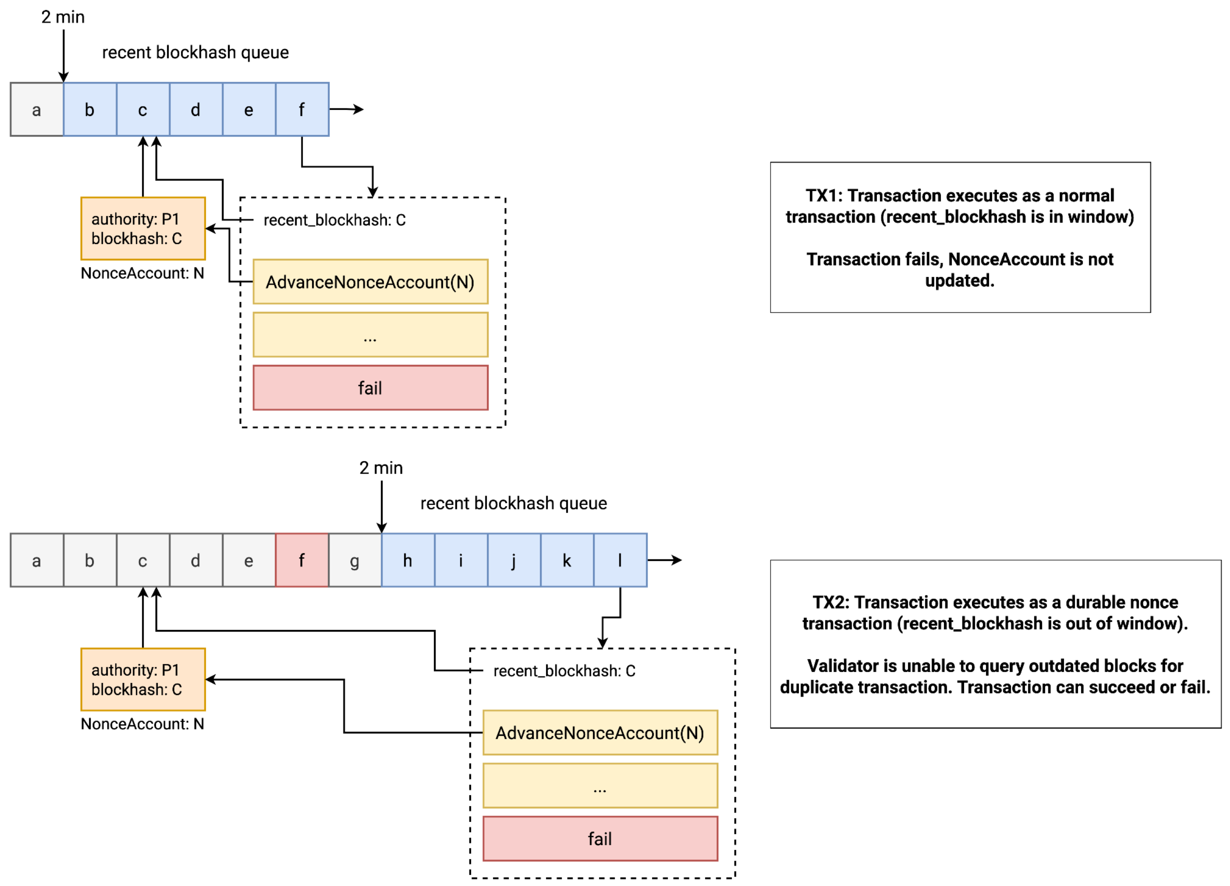
During normal execution, the transaction processor will compare the transaction's `recent_blockhash` to the blockhash stored in the `NonceAccount`. Then the `AdvanceNonceInstruction` will update the blockhash stored in the `NonceAccount` to the current most recent blockhash (thereby "consuming" it). Note that this blockhash update will *always* happen even if the resulting transaction fails as long as the transaction executes as a durable nonce transaction.

Double-execution Bug

Prior to 1.9.28 and 1.10.23 the nonce transaction implementation was vulnerable to a double-execution bug. In some cases, the ability to process this second transaction was dependent on local validator `status_cache` and therefore non-deterministic. On June 1st, 2022 an instance of this bug forked a large portion of validators leading to loss of consensus and the blockchain was halted.

In this section, we outline the three variants of the double-execution bug. These bugs are described in the context prior to the emergency patch.

Variant 1 [patched by #25744]: (fail normal, succeed nonce)

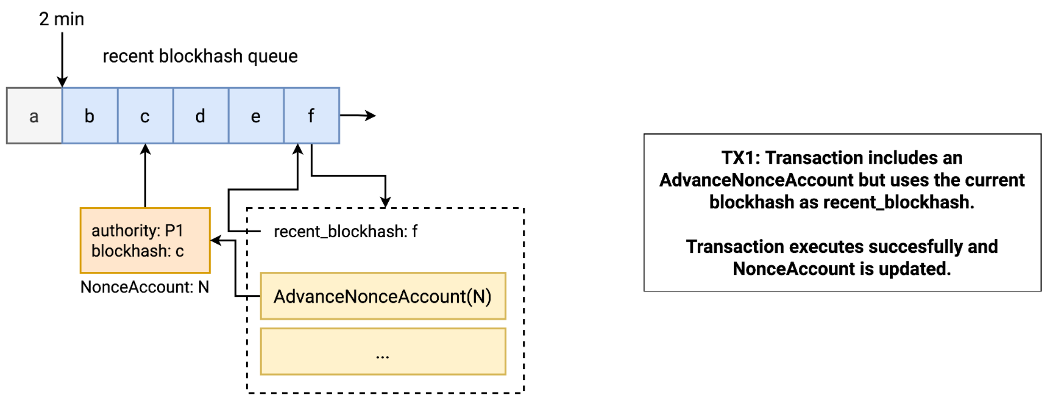

2 min
recent blockhash queue
a b c d e f
authority: P1
blockhash: C
NonceAccount: N
recent_blockhash: C
AdvanceNonceAccount(N)
...
fail
TX1: Transaction executes as a normal transaction (recent_blockhash is in window)
Transaction fails, NonceAccount is not updated.
2 min
recent blockhash queue
a b c d e f g h i j k l
authority: P1
blockhash: C
NonceAccount: N
recent_blockhash: C
AdvanceNonceAccount(N)
...
fail
TX2: Transaction executes as a durable nonce transaction (recent_blockhash is out of window).
Validator is unable to query outdated blocks for duplicate transaction. Transaction can succeed or fail.

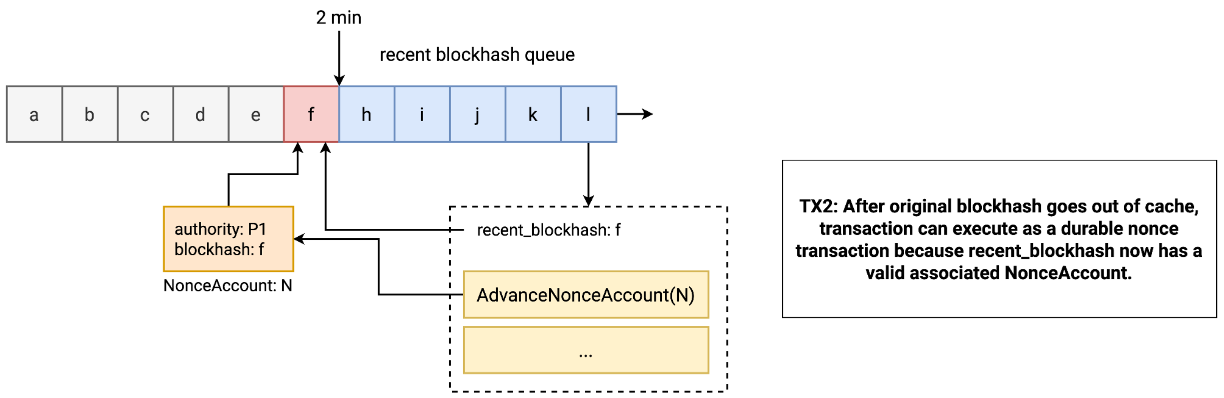
Nonce transactions with a `recent_blockhash` less than 2 minutes old are ambiguous. They can execute as both a regular transaction *and* a nonce transaction. By default, validators will default to a regular transaction if it is possible.

Normally, a nonce transaction that executes as a normal transaction will still invoke its `AdvanceNonceAccount` instruction and therefore de-activate its ability to execute a second time as a nonce transaction (since the associated NonceAccount no longer matches the transaction's recent_blockhash).

However, if the transaction contains an instruction that fails, it will be included in the blockchain but will have its state reverted *including* the NonceAccount state. In normal circumstances, the `NonceAccount` hash would be updated as a safety mechanism to prevent replay even if the resulting transaction fails; however in this case, since the transaction executes as a normal transaction, the update does not occur.

Variant 2 [patched by #25744]: (succeed normal, succeed/fail nonce)


2 min
recent blockhash queue
a
b
c
d
e
f
authority: P1
blockhash: c
NonceAccount: N
recent_blockhash: f
AdvanceNonceAccount(N)
...
TX1: Transaction includes an AdvanceNonceAccount but uses the current blockhash as recent_blockhash.
Transaction executes succesfully and NonceAccount is updated.

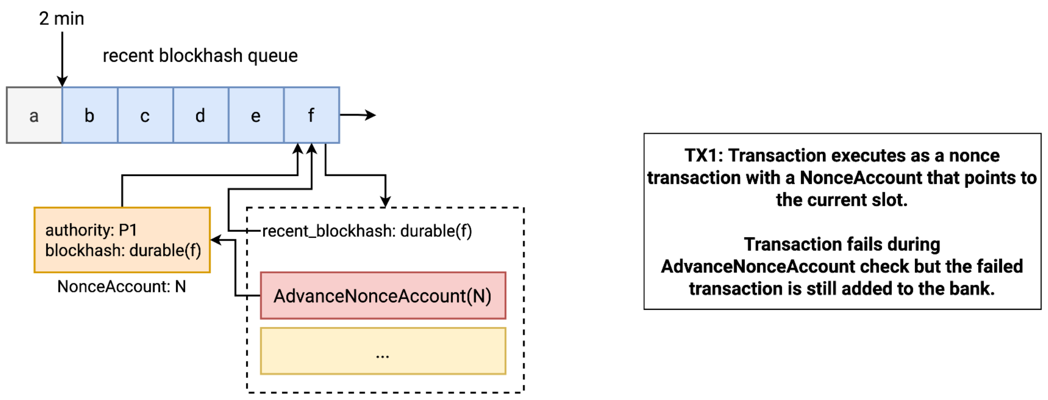

2 min
recent blockhash queue
a
b
c
d
e
f
h
i
j
k
l
authority: P1
blockhash: f
NonceAccount: N
recent_blockhash: f
AdvanceNonceAccount(N)
...
TX2: After original blockhash goes out of cache, transaction can execute as a durable nonce transaction because recent_blockhash now has a valid associated NonceAccount.

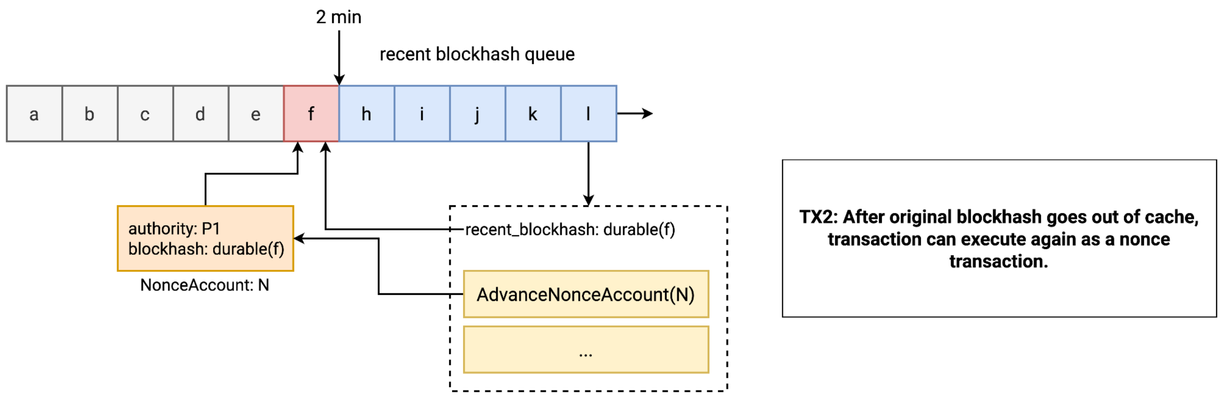
In another case, an invalid nonce transaction (i.e. one which has a `recent_blockhash` that does not match the `NonceAccount`) can first execute as a normal transaction and in the process make the `NonceAccount` valid.

Specifically, if a nonce transaction has a `recent_blockhash` that points to the current most recent blockhash, the initial execution will invoke `AdvanceNonceAccount` which will update the associated `NonceAccount` to have the same stashed blockhash.

Subsequently, when the original slot information is dropped from a validator's status cache, the transaction can be replayed as a nonce transaction. Since the validator status cache is nondeterministic, this can lead to situations where some validators accept the second transaction and others reject it, leading to a possible loss of consensus.

Variant 3 [patched by #25832]: (fail nonce, succeed/fail nonce)


2 min
recent blockhash queue
a
b
c
d
e
f
authority: P1
blockhash: durable(f)
NonceAccount: N
recent_blockhash: durable(f)
AdvanceNonceAccount(N)
...
TX1: Transaction executes as a nonce transaction with a NonceAccount that points to the current slot.
Transaction fails during AdvanceNonceAccount check but the failed transaction is still added to the bank.


2 min
recent blockhash queue
a
b
c
d
e
f
h
i
j
k
l
authority: P1
blockhash: durable(f)
NonceAccount: N
recent_blockhash: durable(f)
AdvanceNonceAccount(N)
...
TX2: After original blockhash goes out of cache, transaction can execute again as a nonce transaction.

In a third variant, transactions can execute twice as a nonce transaction.

This variant is similar to variant #2, but instead, the initial transaction is processed as a nonce transaction due to the changes introduced by #25744.

Specifically, in the case where a nonce transaction has an associated `NonceAccount` that points to the current most recent blockhash, the transaction can initially execute as a nonce transaction. However, `AdvanceNonceAccount` will fail (at the instruction level) because it is unable to advance the nonce. The result is that this failed transaction is included in the slot and the `NonceAccount` is left unchanged; hence, the transaction can execute a second time once the original transaction is removed from the validator status cache.

Bug Impact & Severity

To the best of our knowledge, variant 1 was the only bug type involved in the mainnet outage. However, variant 2 was also possible in the period before the crash. In this section we discuss the potential impact of such a bug. Note that variant 3 is only possible after PR #25744 and was therefore not possible before the crash.

Variant 1

Bug variant 1 can occur through normal usage of a NonceAccount if the original transaction is signed and sent within the 2 minute window (and indeed we suspect this is how it actually happened). However, since the first transaction necessarily fails, the resulting on-chain impact is limited. Effectively, from the on-chain perspective, the result is a single successful transaction.

Variant 2

Bug variant 2 is not likely to happen naturally since it requires explicitly using the NonceAccount in the wrong way–specifically the original transaction needs to contain a blockhash that does not match the NonceAccount. Additionally since the transaction blockhash needs to match the most recent blockhash, a user would need to fetch the current blockhash, sign the transaction, and send it to validators before another slot has been processed (i.e. in less than 600ms at current mainnet speeds). However, when such a situation arises, it is possible to successfully execute the same transaction twice.

While this successful “double-execution” violates certain assumptions about how the system works (namely that transaction executes exactly once), the resulting impact is fairly low considering the circumstances required to trigger this bug:

Firstly, note that unlike a hypothetical Bitcoin “double-spend,” in the Solana model, each successful transaction would be executed in a separate context and the program logic would run twice independently. In this way, while a token transfer instruction could run twice, this bug would not enable an attacker to violate on-chain logic in any programs.

Secondly, any transaction that has the potential to execute twice would need to be specially constructed and all of the original signers would need to acknowledge this (by signing the transaction). It is not possible for an attacker to take an arbitrary transaction and replay it, even ones that use NonceAccounts.

Therefore, we believe there is a low possibility of this bug being used in any sort of exploit against other parties using Solana. Effectively, an attacker would be limited to double-execution of their own, signed transactions which is fundamentally equivalent to simply signing the same transaction twice.

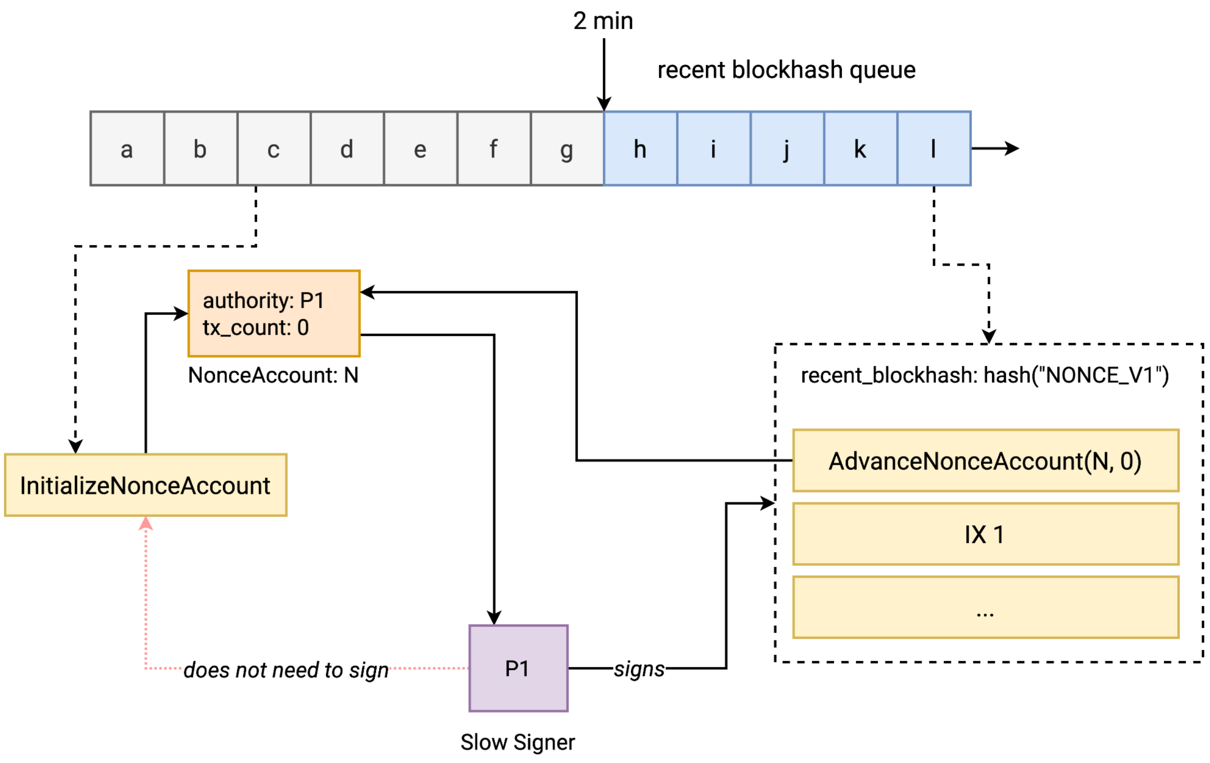
Preventing Footguns

All three bug variants stem from the fact that NonceAccounts use block hashes as a source of unique values:

- Variants 1 and 2 were caused by block hash collisions (between nonce account and blockhash_queue) leading to an ambiguous transaction type.
- Variant 3 was caused by insufficient NonceAccount "consumption," due to the availability of only one unique blockhash per slot.

It is not necessary to use block hashes as a source of unique values. As described in NonceAccount Overview, the required property of a nonce account is that it is only valid for the first transaction that uses it and no subsequent ones.

An alternative, simpler construction could be to create NonceAccounts with a `tx_count` that starts at 0 and increases by 1 each time the `NonceAccount` is used. A nonce transaction that refers to the `NonceAccount` will need to provide the expected `tx_count` parameter to execute successfully. For example:



In this case, the `recent_blockhash` could be set to a fixed, magic value (e.g. `hash(“NONCE_V1”)`) that indicates to the Solana runtime that this transaction should be treated as a nonce transaction (with room to upgrade).

Some mechanism would also need to exist to prevent resetting a NonceAccount and thereby re-enabling old transactions. This mechanism could be integrated on-chain or at the validator level. Additionally, while the existing Solana model prevents cross-cluster replay (due to mutually exclusive blockhash sequences), this model would need to incorporate a feature such as cluster id into the NonceAccount to prevent identical transactions from executing on multiple clusters.

This construction removes many of these footguns and provides some benefits:

- No need for pseudo-blockhashes as implemented in #25744
- Transaction type is clear without needing to check blockhash_queue
- Room for unambiguously upgrading nonce transaction type without backwards compatibility issues
- Advancing a nonce account is always possible (even multiple times per slot)
- Offline signers can queue multiple transactions to use the same nonce account with different expected `tx_count`

03 | Pull Requests

In this section we summarize the changes made by each PR and discuss the effect and security implications.

#25744	Separates durable nonce and blockhash domains
#25832	Reject durable nonce txs that don't use an advanceable nonce
#25788	Permanently disables durable nonces with chain blockhash domain
#25789	Adds system instruction to upgrade legacy nonce versions
#25831	Reject durable nonce txs not signed by authority

#25744: separates durable nonce and blockhash domains

Overview:

- NonceAccount now stores hash(“DURABLE_NONCE”, blockhash) instead of the raw blockhash.
- Durable nonces are now stored in a wrapped `struct DurableNonce(Hash)`

Effect:

- Nonce transactions < 2 minutes old are no longer ambiguous; a durable nonce can never be used as a regular nonce.
- **Fixes bug variants 1 and 2**

Feature Gating:

separate_nonce_from_blockhash Gea3ZkK2N4pHuVZVxWcnAtS6UEDdyumdYt4pFcKjA3ar	Once activated, durable nonce computation will use hashed blockhash instead of raw blockhash.
enable_durable_nonce 4EJQtF2pkRyawwcTVfQutzq4Sa5hRhibF6QAK1QXhtEX	Re-enable durable nonce transactions.

#25832 Reject durable nonce txs that don't use an advanceable nonce

Overview:

- While AdvanceNonceInstruction will fail if the durable nonce is not advanceable, this happens at the instruction level and the resulting transaction can still be included in the blockchain.
- This PR copies the logic to reject nonce transactions whose `NonceAccount` points to the most recent blockhash into core Bank code, thereby failing the whole transaction and not including it.

Effects:

- **Fixes bug variant 3**

Notes:

- Duplicate logic between AdvanceNonceInstruction and Bank to check for advanceable nonces. Internal AdvanceNonceInstruction logic is now redundant.

Feature Gating:

nonce_must_be_advanceable 3u3Er5Vc2jVcwz4xr2GJeSAXT3fAj6ADHZ4BJMZiScFd	When activated, transactions will fail during `Bank::check_transaction_for_nonce` if the NonceAccount points to the current most recent blockhash.

#25788: permanently disables durable nonces with chain blockhash domain

Overview:

- Old NonceAccounts have type `Versions::Legacy`, new NonceAccounts (with hashed durable nonce) have type `Versions::Current`
- Old NonceAccounts are prevented from being used in nonce transactions

Effect:

- All old NonceAccounts are disabled.
- This prevents cases where a transaction executed successfully before the crash and is now a valid NonceTransaction.

Notes:

- Syntax is a bit confusing, several calls (e.g. `Versions::new`) require inline comments to explain arguments.

Feature Gating:

Implicitly gated by **enable_durable_nonce** from #25744.

#25789 adds system instruction to upgrade legacy nonce versions

Overview

- Adds `SystemInstruction::UpgradeNonceAccount`.
- Allows legacy nonce accounts whose nonce state refers to a raw blockhash to upgrade to the new format as specified in #25744 and mandated by #25788

Effect:

- The UpgradeNonceAccount instruction converts a Legacy NonceAccount into a Current NonceAccount with the durable nonce set to the equivalent DurableNonce for the NonceAccount's blockhash.

Notes:

- This instruction is redundant; `SystemInstruction::AdvanceNonceAccount` implicitly will convert Legacy NonceAccounts into Current NonceAccounts, thereby enabling their reuse.
- The motivation of a `memo` argument in UpgradeNonceAccount is not clear.

Feature Gating:

Implicitly gated by **enable_durable_nonce** from #25744.

#25831 Reject durable nonce transactions not signed by authority

Overview:

- Solana durable nonce transactions must now be signed by the nonce authority.
- Previously, this was validated at the instruction level in `AdvanceNonceAccount`.
    - Durable nonce transactions not signed by the nonce account's authority could still make it into a block, and then fail at program runtime.

Effects:

- Nonce transactions without the appropriate authority's signature are now immediately rejected before being added to a block

Feature gating:

nonce_must_be_authorized HxrEu1gXuH7iD3Puua1ohd5n4iUKJyFNtNxk9DVJkvgr	When activated, transactions will fail during `Bank::check_transaction_for_nonce` if the signer for the NonceAccount is not provided.

04 | General Suggestions

In this section we include some general suggestions related to durable nonce implementation..

OS-SOL-DN-SUG-01	Consider a simplified NonceAccount model
OS-SOL-DN-SUG-02	Standardize location of nonce transaction processing
OS-SOL-DN-SUG-03	Remove UpgradeNonceAccount
OS-SOL-DN-SUG-04	Clarify NonceAccount versioning syntax
OS-SOL-DN-SUG-05	Ensure features are activated simultaneously

OS-SOL-DN-SUG-01 **[Info]**: Consider a simplified NonceAccount model

Description

The three bugs related to durable nonce transactions stem from the usage of blockhash in the NonceAccount. This usage is not required as described in Preventing Footguns.

OS-SOL-DN-SUG-02 [Info]: Standardize location of nonce transaction processing

Description

Currently there is a lot of duplicated logic between Bank code and internal instruction code (i.e. AdvanceNonceAccount). Processing a nonce transaction requires a mix of standard internal instructions (AdvanceNonceAccount) and "magic" external functionality in Bank.

This duplication can lead to confusion and edge cases. For example, if AdvanceNonceAccount fails, the transaction may still be added to the chain. Several of these PRs (#25832, #25831) simply duplicate internal logic into the outer bank processing code to prevent failed nonce transactions from being added to the blockchain.

Consider standardizing all nonce transaction processing in Bank rather than relying on internal instruction processing.

OS-SOL-DN-SUG-03 **[Info]**: Remove UpgradeNonceAccount

Description

UpgradeNonceAccount as introduced in #25789 is redundant as AdvanceNonceAccount can already upgrade Legacy NonceAccounts automatically.

Consider removing this instruction to prevent bloat and/or confusion.

OS-SOL-DN-SUG-04 [Info]: Clarify NonceAccount versioning syntax

Description

Introduced in #25788, NonceAccounts now have both a Legacy and a Current version. The current terminology/syntax is confusing and requires inline comments to explain mystery arguments for example in `Versions::new`.

Cleaning up this syntax and possibly restructuring NonceAccount version structures could reduce confusion, improve readability, and help prevent future bugs.

OS-SOL-DN-SUG-05 [Info]: Ensure features are activated simultaneously

Description

Activating **separate_nonce_from_blockhash** before **nonce_must_be_advanceable** will reintroduce bug variant 3.

Therefore, it is critical that these features are activated simultaneously.