Background

The Arbitrum challenge protocol, which is run in the rare case where validators are staked on conflicting claims about the chain history, creates a “chess clock” for each side in a challenge. The chess clock is a total time allocation for that side to make its moves. The chess clock is running when it is that side’s turn to move, and if the chess clock runs out the side forfeits the challenge.

Each move in the challenge is a transaction on the parent chain (i.e. Ethereum for an L2 chain’s challenge protocol, and an L2 chain for an L3 chain’s challenge protocol). The chess clock is given enough time so that a side can make all of its moves in the challenge, even in the presence of a worst-case censorship attack against the parent chain.

The issue

A question that arises for L3 Orbit chains is: what is the worst-case censorship attack against the underlying L2 chain, such as Arbitrum One?

The worst case happens if the L2 sequencer is malicious, and tries to censor every move made by one side in the challenge. Of course the L2 sequencer can’t censor a transaction forever, but it can delay the transaction’s inclusion by the force-inclusion delay (currently 24 hours, though there are proposals to cut this to, say, 8 hours). A transaction that has been waiting in the delayed inbox for the force-inclusion delay (or longer) can be forced into the sequence whether the sequencer likes it or not. (Anyone can force inclusion of such a message, by making a special call to the inbox contracts that live on L1 Ethereum.)

To cause this kind of delay for a transaction, the L2 sequencer would have to refuse to include the transaction in its sequence, then refuse to include any transactions submitted through the delayed inbox, which means not including any asset deposits to the L2 chain, thereby requiring a forced inclusion after the 24 hour delay.

While this is less than the 7 day maximum that is commonly assumed for censorship on Ethereum—that’s the reason for the 7 day challenge period in Arbitrum One’s settlement protocol—the difference is that in theory the L2 sequencer could cause this delay repeatedly, for every move by one side in the challenge game, adding up to many days of delay.

How to fix it

Fortunately this issue is not too hard to fix. We simply need to prevent the L2 sequencer from carrying out a worst-case censorship attack repeatedly.

The Arbitrum Security Council already has the power to fire the L2 sequencer and appoint a replacement, so in the short to medium term, we can rely on the Security Council to replace the L2 sequencer if it tries to do repeated long censorship attacks. This is a reasonable assumption because these attacks would affect many users of the L2.

In the longer run, the sequencer protocol can be changed to prevent repeated 24-hour (or whatever) delays. For example, whenever an abnormal delay of length D

occurred, the protocol could subtract D/2

from the maximum delay parameter going forward. This would guarantee that a series of delays could add up to no more than double the initial force-inclusion delay. (The subtracted-off amount could be added back slowly afterward, so a well-behaved sequencer would get back its safety margin.) Other approaches are possible, so presumably the DAO will debate which approach is best.

For now, it seems safe to assume that the Security Council will replace the sequencer if it tries to commit repeated censorship of time-critical transactions.

Appendix: Wait, why is the force-inclusion delay 24 hours anyway?

You might be wondering why the force-inclusion delay is 24 hours. Why not allow a neglected transaction to be forced into the sequence sooner?

The answer is that forcing inclusion of a transaction typically causes a reorg of the L2 chain, and Arbitrum chains are designed to almost never reorg—that’s why most people decide to trust the sequencer’s immediate transaction result. Over the history of Arbitrum One, there has been only one reorg, affecting a handful of transactions.

So the force-inclusion delay is currently set to be longer than the reasonable worst-case sequencer downtime. The current sequencer uses a highly redundant devops setup, with multiple hot spares in diverse locations, but that wouldn’t prevent downtime if there is some bug in the sequencer that causes it to get into a stuck state that can’t be fixed by rebooting. In that case, engineers would need to diagnose and fix the bug before the sequencer could resume.

Under normal conditions, the force-inclusion delay doesn’t matter because an honest sequencer will include delayed-inbox messages once they have finality on Ethereum, with a delay on the order of ten minutes. So the initial judgment was that preventing reorgs was important relative to the worst-case inclusion delay that might result from a serious bug.