```
BIP: 50
Title: March 2013 Chain Fork Post-Mortem
Author: Gavin Andresen <gavinandresen@gmail.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0050
Status: Final
Type: Informational
Created: 2013-03-20
License: PD
```

What went wrong

A block that had a larger number of total transaction inputs than previously seen was mined and broadcasted. Bitcoin 0.8 nodes were able to handle this, but some pre-0.8 Bitcoin nodes rejected it, causing an unexpected fork of the blockchain. The pre-0.8-incompatible chain (from here on, the 0.8 chain) at that point had around 60% of the mining hash power ensuring the split did not automatically resolve (as would have occurred if the pre-0.8 chain outpaced the 0.8 chain in total work, forcing 0.8 nodes to reorganise to the pre-0.8 chain).

In order to restore a canonical chain as soon as possible, BTCGuild and Slush downgraded their Bitcoin 0.8 nodes to 0.7 so their pools would also reject the larger block. This placed majority hashpower on the chain without the larger block, thus eventually causing the 0.8 nodes to reorganise to the pre-0.8 chain.

During this time there was at least one large double spend. However, it was done by someone experimenting to see if it was possible and was not intended to be malicious.

What went right

- The split was detected very quickly.
- The right people were online and available in IRC or could be contacted directly.
- Marek Palatinus (Slush) and Michael Marsee (Eleuthria of BTCGuild) quickly downgraded their nodes to restore a pre-0.8 chain as canonical, despite the fact that this caused them to sacrifice significant amounts of money.
- Deposits to the major exchanges and payments via BitPay were also suspended (and then un-suspended) very quickly.
- Fortunately, the only attack on a merchant was done by someone who was not intending to actually steal money.

Root cause

Bitcoin versions prior to 0.8 configure an insufficient number of Berkeley DB locks to process large but otherwise valid blocks. Berkeley DB locks have to be

manually configured by API users depending on anticipated load. The manual says this:

> The recommended algorithm for selecting the maximum number of locks, lockers, and lock objects is to run the application under stressful conditions and then review the lock system's statistics to determine the maximum number of locks, lockers, and lock objects that were used. Then, double these values for safety.

With the insufficiently high BDB lock configuration, it implicitly had become a network consensus rule determining block validity (albeit an inconsistent and unsafe rule, since the lock usage could vary from node to node).

Because max-sized blocks had been successfully processed on the testnet, it did not occur to anyone that there could be blocks that were smaller but require more locks than were available. Prior to 0.7 unmodified mining nodes self-imposed a maximum block size of 500,000 bytes, which further prevented this case from being triggered. 0.7 made the target size configurable and miners had been encouraged to increase this target in the week prior to the incident.

Bitcoin 0.8 did not use Berkeley DB. It switched to LevelDB instead, which did not implement the same locking limits as BDB. Therefore it was able to process the forking block successfully.

Note that BDB locks are also required during processing of re-organizations. Versions prior to 0.8 may be unable to process some valid re-orgs.

This would be an issue even if the entire network was running version 0.7.2. It is theoretically possible for one 0.7.2 node to create a block that others are unable to validate, or for 0.7.2 nodes to create block re-orgs that peers cannot validate, because the contents of each node's blkindex.dat database is not identical, and the number of locks required depends on the exact arrangement of the blkindex.dat on disk (locks are acquired per-page).

Action items

Immediately

Done: Release a version 0.8.1, forked directly from 0.8.0, that, for the next two months has the following new rules:

1. Reject blocks that would probably cause more than 10,000 locks to be taken.
2. Limit the maximum block-size created to 500,000 bytes
3. Release a patch for older versions that implements the same rules, but also increases the maximum number of locks to 537,000
4. Create a web page on bitcoin.org that will urge users to upgrade to 0.8.1, but will tell them how to set DB_CONFIG to 537,000 locks if they absolutely cannot.

5. Over the next 2 months, send a series of alerts to users of older versions, pointing to the web page.

Alert system

Done: Review who has access to the alert system keys, make sure they all have contact information for each other, and get good timezone overlap by people with access to the keys.

Done: Implement a new bitcoind feature so services can get timely notification of alerts: -alertnotify= Run command when an AppliesToMe() alert is received.

Done: Pre-generate 52 test alerts, and set a time every week when they are broadcast on -testnet (so -alertnotify scripts can be tested in as-close-to-real-world conditions as possible).

Idea from Michael Gronager: encourage merchants/exchanges (and maybe pools) to run new code behind a bitcoind running the network-majority version.

Safe mode

Done: Perhaps trigger an alert if there is a long enough side chain detected, even if it is not the main chain. Pools could use this to automatically suspend payouts if a long side-chain suddenly appeared out of nowhere (it's hard for an attacker to mine such a thing).

Testing

Start running bots on the testnet that grab some coins from a testnet faucet, generate large numbers of random transactions that split/recombine them and then send them back to the faucet. Randomized online testing on the testnet might have revealed the pathological block type earlier.

Double spending

A double spend attack was successful, despite that both sides of the chain heard about the transactions in the same order. The reason is most likely that the memory pools were cleared when the mining pool nodes were downgraded. A solution is for nodes to sync their mempools to each other at startup, however, this requires a memory pool expiry policy to be implemented as currently node restarts are the only way for unconfirmed transactions to be evicted from the system.

Resolution

On 16 August, 2013 block 252,451 (0x0000000000000024b58eeb1134432f00497a6a860412996e7a260f47126eed07) was accepted by the main network, forking unpatched nodes off the network.

Copyright