Bitcoin Transaction Malleability in 2018

Will McChesney | Alok Menghrajani

Tl;dr: Transaction malleability used to be a potential risk when transacting on the Bitcoin network. Today, legacy transactions are protected from malleability by ad-hoc checks. Until the adoption of Segregated Witness, miners have the ability to perform some forms of malleability abuse.

Bitcoin transactions have a transaction id (txid) formed as a hash over the data involved in the transaction. It is common for wallet related software to assume that this identifier is immutable.

However, the txid is only immutable once the exact data in the transaction has been finalized by being mined into the blockchain. Until then, any node on the network is able to make subtle changes to the underlying data, which results in a different hash.

This is not a security issue because it is not possible to alter how many bitcoins are transferred from what input to what output. However, details such as the signature "encoding" or the exact signature script can be altered: they are malleable, making the txid malleable as well.

Transaction malleability has been a known issue for a long time (since at least 2011). A proposal to fix this issue (BIP 62, Dealing with malleability) using ad-hoc checks was authored by Pieter Wuille in March 2014. The proposal was withdrawn in November 2015. The fact that the proposal was withdrawn does not imply that transaction malleability is easy to perform or still a major risk.

This document collects the history of transaction malleability and when various fixes were implemented. We link to relevant commits and code snippets in Bitcoin Core's GitHub repo.

In the longer term, Segregated Witness (segwit) solves transaction malleability in a cleaner way: by not including the signature in the txid computation.

Transaction validation

A transaction specifies inputs and outputs. The inputs of a given transaction refer to outputs from previous transactions with an additional script fragment (scriptSig). The outputs of a given transaction are script fragments that define a spending condition (scriptPubKey).

For a transaction to be valid, the spending condition from previous outputs must correctly validate. This validation happens by concatenating scriptSig and scriptPubKey and evaluating the resulting script in an interpreter. If the remaining value on the evaluation stack is true, the spending condition is valid.

In the transaction malleability case, we look at ways to mutate scriptSig without changing the interpreter's final result.

Non-push operations in scriptSig

Typically, a scriptSig only contains push operations. Non push operations enable the creation of equivalent scripts in an infinite number of ways. For example, pushing 1, pushing 2 and then performing an addition operation is equivalent to pushing 3.

Gavin Andresen implemented IsPushOnly as part of standard transaction validation in December 2010 (a206a2). By default, this check is enabled in mainnet and disabled in testnet.

It seems Pieter Wuille added additional code to perform a similar check in October 2014 (d752ba). Pieter's change is however only enabled in tests. Did he forget to add SCRIPT_VERIFY_SIGPUSHONLY to the standard flags?

Fun quirk: IsPushOnly() considers OP_RESERVED to be a push-type opcode (the code simply checks if the opcode is less than a specific value).

Push operations in scriptSig of non-standard size type

The scripting language uses variable length encoding to save space. It was possible to encode push values in various equivalent ways.

Pieter Wuille added CheckMinimalPush in October 2014 (698c6a).

Superfluous scriptSig operations

Only the top element of the stack needed to be true for a script evaluation to be considered valid. Superfluous values could remain below the top element.

Pieter Wuille implemented a check in October 2014 (added in b6e03c and enabled in da918a) to ensure the stack only contains a single element. This check only applies to some types of transactions.

Non-DER encoded ECDSA signatures

It was possible to encode signatures in different formats. This was possible, because the bitcoin code was passing the data to OpenSSL, which accepts various different formats (DER, BER, etc.).

Today, IsValidSignatureEncoding enforces that the signature is encoded in a canonical way.

The function was initially named IsDERSignature and was implemented by Pieter Wuille in October 2014 (9df9cf) as part of BIP 62.

IsDERSignature was renamed to IsValidSignatureEncoding (80ad13) as part of BIP 66 (Strict DER signatures).

Low S

Given an ECDSA signature (r, s), the signature (r, -s) is also valid because the underlying curve is symmetric.

An initial proposal was to enforce that s is even. Pieter Wuille implemented the low S check in February 2014 (6fd7ef) and Gregory Maxwell enabled the check in October 2015 (b196b6) and the following commit message: "[...]If widely deployed this change would eliminate the last remaining known vector for nuisance malleability on boring SIGHASH_ALL p2pkh transactions."

The low S check was proposed in BIP 146 (Dealing with signature encoding malleability).

Conclusion

- The majority of items mentioned in BIP 62 were implemented before the proposal was withdrawn. The remaining items don't apply from the perspective of an external node and typical transactions.
- We were initially confused by the fact that nodes use different rules to relay new transactions vs to validate transactions in the blockchain.
- We were able to mutate transactions on the testnet. None of the published malleability vectors work on mainnet.
- "Unexpected" transactions put wallet software in inconsistent states. User's funds are locked until they manually fix the wallet.
- At this point in time, we believe the main risks around transaction malleability are:
    - Someone discovering a new vector. The most likely place to look for this would be an implementation gap in the existing checks or a new issue similar to Low S.
    - A miner behaving poorly. Given the high difficulty to mine blocks this is not a big concern.

Links

- Chechik D., Hayak B.
  Bitcoin Transaction Malleability Theory In Practice (BlackHat 2014)
- Decker C., Wattenhofer R. (2014)
  Bitcoin Transaction Malleability and MtGox
- Andrychowicz M., Dziembowski S., Malinowski D., Mazurek Ł. (2015)
  On the Malleability of Bitcoin Transactions
- Bitcoin Forum and Bitcoin dev mailing list