Leaf-Node Vulnerability in Bitcoin Merkle Tree Design

Date: August 4th, 2017

Author: Sergio Demian Lerner, RSK Labs

Introduction

This document describes a vulnerability in Bitcoin Design that reduces the security of SPV proofs. An attacker can create a valid SPV proof for a fake payment to a victim, for an arbitrary number of bitcoins, and trick the victim into accepting this payment as valid. Happily, exploiting this bug requires brute-forcing between 69 and 73 bits (depending on initial investment), each operation being a double SHA2. For example, an attack can be carried on with an investment of 3M USD. It is assumed that most SPV wallets will be vulnerable to this attack. Also vulnerable are automated systems that accept SPV proofs (such as the Elements blockchain or the RSK Bridge contract). A simple patch which it neither a soft nor a hard fork can prevent the attack described. Also, a second attack that forks the Bitcoin blockchain is presented, requiring the brute-forcing of 225 bits, so it's only of theoretical interest.

The Problem

Bitcoin Merkle tree makes no distinction between inner nodes and leaf nodes. The depth of the tree is implicitly given by the number of transactions. Inner nodes have no specific format, and are 64 bytes in length. Therefore, an attacker can submit to the blockchain a transaction that has exactly 64 bytes in length and then force a victim system to re-interpret this transaction as an inner node of the tree. An attacker can therefore provide an SPV proof (a Merkle branch) that adds an additional leaf node extending the dual transaction/node and provide proof of payment of any transaction he wishes.

It must be noted that a problem reciprocal to this was identified by Andrew Miller [1]. He realized that internal nodes could be re-interpreted as transactions, but it seems he didn't publish (or didn't discover) the opposite attack: transactions re-interpreted as nodes.

Crafting a Transaction-node in 2^72 operations

The following diagram shows a 64-byte Bitcoin transaction, and how this transaction is split into two 32-byte chunks.

Absolute Offs	32-byte offs	Size	Description	Brute force bits (stage)
0	0	4	Version	0
4	4	1	Input count	0
5	5	27	Input 0 Tx hash (part 1)	0
32	0	5	Input 0 Tx hash (part 2)	40 (2)
37	5	4	Input 0 Txout index	17 of 32 (1)
41	9	1	Input 0 script length	8 (1)
42	10	0	Empty Script	0
42	10	4	nSequence	0 (anything allowed)
46	14	1	Output count	8 (1)
47	15	1	Output 0 script length	8 (1)
48	16	4	Output 0 scriptPubKey	0 (anything allowed)
52	20	8	value	29 of 64 (1)
60	28	4	lock_time	2 of 32 (1)
				Total (1) = 72 Total (2) = 40

Let the transaction to become an inner node be T. The brute-forcing is done in two stages. Suppose that the attacker holds 2^36-1 satoshis (= 687 BTC or about 1.9M USD) in a single UTXO A.

First Brute-forcing stage

First the attacker proceeds with a second half of T. In the second half, some fields are fixed, some free and some partially free. The LockTime value is also partially brute-forced: the attacker makes sure the brute-forced uint32 is between 500000000 and 1501843940 (today as a Unix timestamp) or between 0 and 479042 (the current block height). This implies that the LockTime has elapsed. The combined numerical range amplitude is 1002322982, which is close to 2^{30}. Therefore, he only needs to bruteforce 2 bits of the LockTime. The higher bits of the Previn tx output index are brute-forced so the index is always less than 32768.

The value is only partially brute-forced. From the 64 field bits, 35 bits are chosen freely (because the attacker has 2^36-1 satoshis, so 35 free bits can represent any number lower than the amount), while the 29 MSB bits must be zero. If the attacker has more BTC, he can reduce the number of bits to brute-force by consuming more BTC in A. Note that the amount is not lost: as the attacker is a miner, he can collect the transaction fees and consume the transaction output and recover all the funds (however creating the anyone-can-spend output and a high-fee transaction T will put the attacker in the risk that another miner reverts the blockchain to re-mine this block and collect both fee or output in T). The attacker may mine several blocks in-private (selfish-mining) to prevent rollbacks.

The attacker tries to create a fake transaction F whose transaction ID matches this second half of T. An easy way to do it is to create a fake transaction having one output for the fake payment to the victim, and scan all possible values for the lock_time field; when the space is exhausted, the input script is slightly modified and a new 32-bit space of lock_time values is created. Again, an AsicBoost-like technique can be used to save one message-scheduler of the double-SHA2 operation.

The total number of bits to brute-force in the first stage is therefore 72 bits.

Second Brute-forcing stage

In this second stage, the attacker tries to brute force the first 32-byte half of T. Let Q be the tx output index chosen in the first stage. He creates a huge number of transactions consuming the input A and having Q outputs each. The last output will be consumed by T. The attacker makes small changes to the tail of the transaction and re-hashes to obtain the id, therefore creating new transaction ids requires only two SHA2 compressions (one to finalize the transaction hash and the double-hash). The lock_field can be used to iterate. The attacker tries to find a transaction whose transaction ids end in the 5-bytes tail chosen in the first stage. Let's call this transaction P. This transaction consumes an output controlled by the attacker having A funds. Approximately 2^40 operations will need to be performed to find P. An AsicBoost-like technique can be used to save one message-scheduler of the double-SHA2 operation.

The total number of bits to brute-force in the first stage is therefore 40 bits.

Interestingly, the brute-forced half of the transaction created in the first stage can be reused as many times if the second stage is re-done. The attacks can be carried out at different times, in different blocks. Therefore, following attacks require only brute-forcing 40 bit each. If the number of outputs in P is 33K, the amortized cost of the attack corresponds approximately to brute-forcing only 65 bits. A final transaction E is added in the same block to consume the output of T and recover the funds at risk.

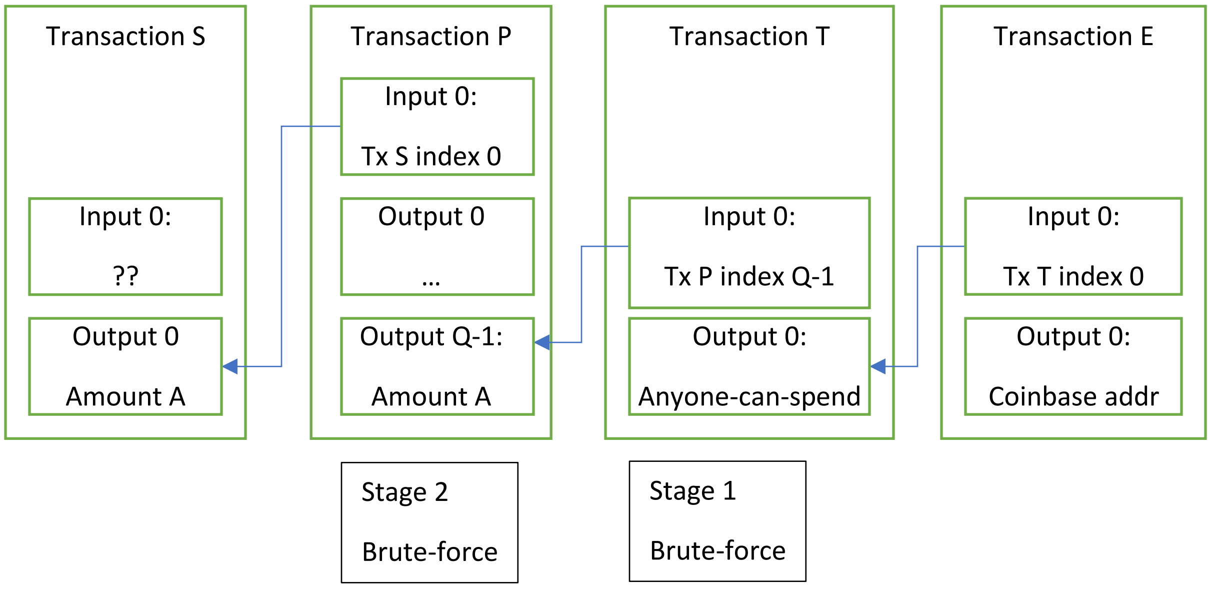

Fig 1. Chain of transactions in attacker's block

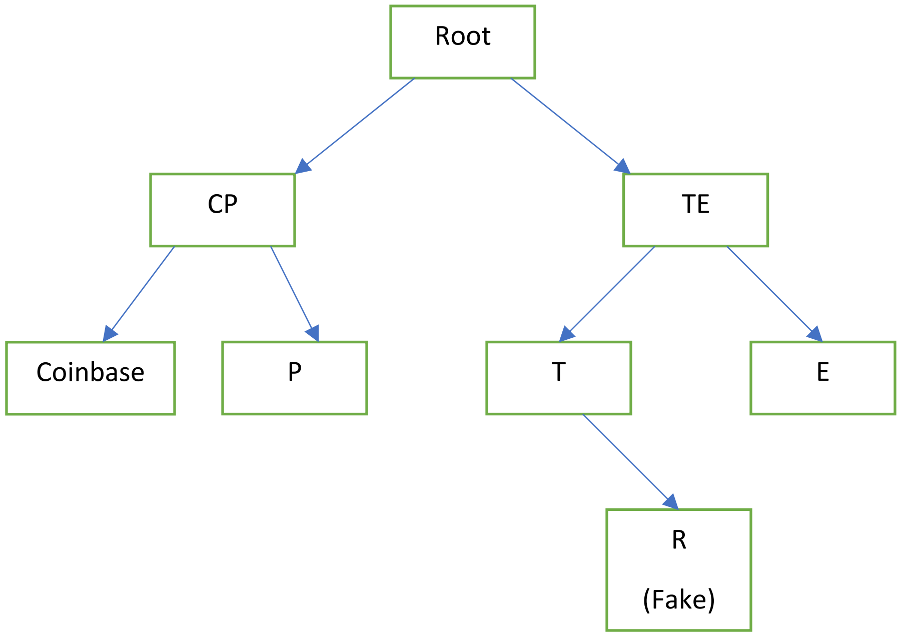

Fig 2. Merkle Tree of the Block containing the fake transaction R

The Cost of the Attack

The technology required to build a custom ASIC that performs the brute-forcing of the second round is very similar to the technology used for Bitcoin miners.

A state-of-the-art Bitcoin miner reaches 14 TH/s and costs $2100 USD. An attacker that buys 1000 units, investing 2.1M USD in hardware, can scan a 72-bit space in 4 days. An additional 5M-10M USD for ASIC chip design and tape-out may be required. If a chip already exists, this cost is saved.

Mining several blocks in private to confirm the fake transaction may be needed to prevent the blockchain to be reorganized by other miners to steal the fees of transaction T. If the attacker is not in collusion with 51% of the miners, then this may cost the attacker millions in rented hashing power, if available. If the attacker is colluding with 51% of the miners there is no additional cost.

Therefore, under favorable conditions to the attacker, the attack can profitable if the attacker can cheat one or more users for 2.1M USD or more in total. As any rational actor receiving s large amount of BTC will double-check the reception using a full node, only autonomous systems relying on SPV proofs (such as the Elements blockchain and the RSK bridge) may suffer from these attacks.

The attack presented in this document is not optimal: there are several trade-offs than can be made to reduce brute-forcing increasing other variables. Locking twice the money in A allows the attacker to reduce the cost in mining hardware to a half, etc.

An (expensive) attack to partition Bitcoin

The same vulnerability allows an attacker to fork the Bitcoin blockchain in two without reconciliation, however this attack costs 2^240 double SHA2 operations, so the attack is impractical. The attack requires the attacker to mine a block A with only one specially crafted coinbase transaction of 64 bytes, and create a competing block B with 2 transactions where the pair of transactions IDs in B hashes to the coinbase transaction in A. Both blocks are broadcast simultaneously to different parts of the Bitcoin network. Therefore approximately 50% of the network receives A and the other 50%, B. The two sets of transactions in A and B are created so that they consume different UTXOs and/or create different UTXOs, and therefore both blockchains are valid, but incompatible.

The fact that the first transaction is the coinbase transaction, and a coinbase transaction requires 27 zero bytes and 1 fixed byte in the first 32 bytes, makes the attack practically unfeasible, as it requires brute-forcing 225 bits.

Remedy

One easy remedy is that SPV wallets check that every internal 64-bit node of the SPV proof is not a valid transaction. The probability of this occurring by random is sufficiently low, so no damage is done. There is no need to hard-fork or soft-fork Bitcoin, although a future programmed hard-fork could fix this vulnerability by adding a prefix to internal nodes before hashing and adding a different prefix to transactions and inner nodes before hashing. Ripple ([2]) uses a prefix system.

Thanks

Special thanks to Juliano Rizzo and Matias Marquez from RSK Labs R&D for helping me reduce the attack complexity.

[1] https://cs.umd.edu/~amiller/BTCRelayAudit.pdf

[2] https://wiki.ripple.com/Hash_Tree