Depressing the Crypto Economy with DoS Bugs

Aleks Kircanski

CackalackyCon, June, 2019

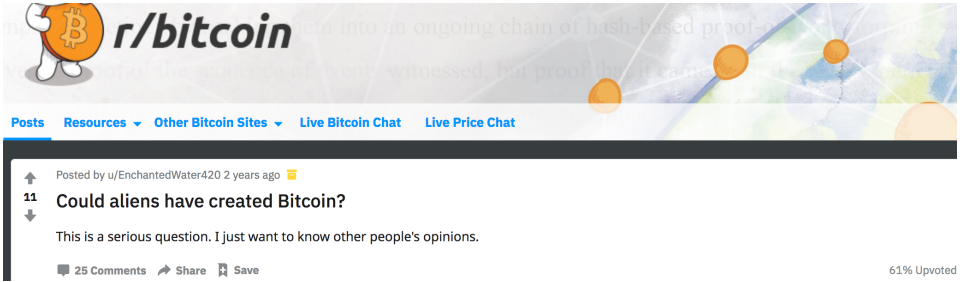
nccgroup

What this talk is about

- Agenda
  - Overview of the Satoshi blockchain client
  - Most common security issue types
  - Netsplit/fork attacks
  - Application-level DoS issues
- Not covered in this talk
  - Smart contract hacking
  - Privacy preserving crypto
  - Alternative consensus algorithms and scalable blockchain
- Goal
  - Provide an idea what basic blockchain vulnerabilities look like
  - ...if you want to audit cryptocurrency software yourself
  - ...WCGW if I try to reimplement Satoshi's client?

nccgroup

Bitcoin and derivatives



- Oct 2008: Bitcoin paper
- Electronic cash without a central authority or financial institutions
- Decentralized system? P2P network
- Main question:
  - How can nodes establish a synchronized view of the ledger?
- Main novel idea in Bitcoin:
  - Establish consensus via the Proof of Work concept

Bitcoin and derivatives

The Bitcoin community:

- Maintained and refactored the code
- Changed existing features and added new ones
- Uncovered and fixed a number of important security issues

Auditing a blockchain client?

- ...or implementing one from scratch?

It makes sense to:

- Learn from the past mistakes
- Rely on previous Bitcoin community's effort
- Avoid the same pitfalls

Blockchain rough overview

Permissioned/permissionless:

- Permissioned: Enterprise blockchain solutions
- Often times centralized systems with some accountability
- In fact: usually, not Satoshi blockchain at all

Smart contract language capabilities:

- Turing complete language
- Less expressive smart contract language

Consensus algorithm:

- Proof of Work (PoW) or Proof of Stake (PoS)
- Adapations on BFT algorithms

Privacy preserving?

- Zero knowledge proof based
- Ring signature based

Security researchers' attention span vs. blockchain

Permissioned blockchain

- Many of these projects aren't open source
- Security of these systems not well understood

Smart contract hacking

- Lots of companies offer smart contract review
- Static analysis tools

Consensus implementation issues

- Arguably under-researched by bug hunters
- BFT implementations vary in large degree

Blockchain client

Some things a blockchain client does:

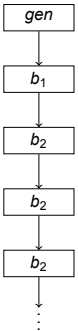
- Listens on the network for new protocol messages
- Ingests transactions and blocks
- Validates txs and blocks
- Stores a copy of the ledger
- Attempts to mine new blocks
- Attempts to be in sync with other nodes

WCGW What could go wrong

Blockchain client state

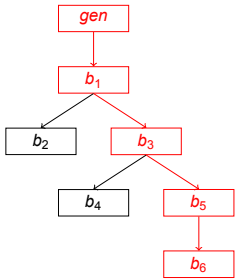
- Nodes' state: ledger
- Ledger: who owns what
- Block: groups transcations together
- Transaction: unlocks existing coins
- Each block refers to a unique previous block
- Each tx refers to one or more prev. txs

Ledger: UTXO (unspent transactions outputs)



Blockchain client state

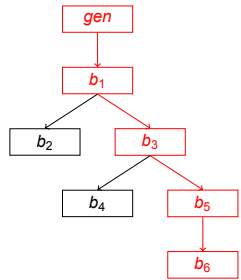
- Block-tree
- ...all the possible ledgers
- Anyone can post blocks on the network
- Ledger: a branch in the tree
  - call it *active* branch
- Suppose all nodes see the same block tree
  - ...assuming good network conditions
  - ...no major discrepancy between nodes



Blockchain client state

Active branch criterion?

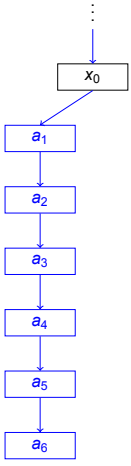
- Come up with an active branch criterion
  - Criterion resistant to manipulation
  - Switching branches should be controlled
- Prevent a *double-spend* attack
  - Spend on one branch
  - Have the system switch to another branch

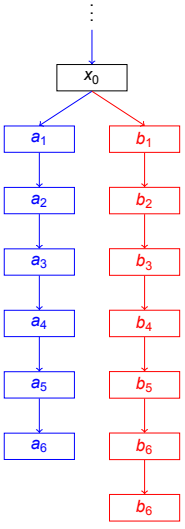


Double-spend attack

Assume: active branch is the longest branch

- block x_0: tx broadcasted
- block a_1: 1 confirmation
- block a_6: tx confirmed
- a miner publishes branch *b*
- branch *b* contains the double-spend tx





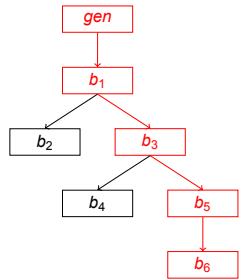
Blockchain client state

What are we trying to do?

- All nodes see the same block tree
- Active branch?
- ...such that switching branches difficult

Bitcoin idea: rely on a PoW puzzle

- Solution easy to verify
- Difficult to solve



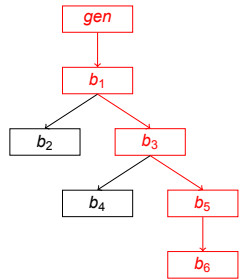
Blockchain client state

Plug in the PoW concept as follows

- Add a block? Solve the PoW puzzle
- Active branch? A branch with most work

Switching branches now:

- Requires computational power
- Security depends on the network's hash power



Bitcoin CVEs

Page Discussion Read View source View history Search Bitcoin Wiki

Common Vulnerabilities and Exposures

CVE	Announced	Affects	Severity	Attack is...	Flaw	Net
Pre-BIP protocol changes	n/a	All Bitcoin clients	Netsplit[1]	Implicit[2]	Various hardforks and softforks	100%
CVE-2010-5137	2010-07-28	wxBitcoin and bitcoind	DoS[3]	Easy	OP_LSHIFT crash	100%
CVE-2010-5141	2010-07-28	wxBitcoin and bitcoind	Theft[4]	Easy	OP_RETURN could be used to spend any output.	100%
CVE-2010-5138	2010-07-29	wxBitcoin and bitcoind	DoS[3]	Easy	Unlimited SigOp DoS	100%
CVE-2010-5139	2010-08-15	wxBitcoin and bitcoind	Inflation[5]	Easy	Combined output overflow	100%

Around 30 CVEs:

- DoS: 14
- Netsplit: 4
- Theft: 4
- Exposure: 4
- Inflation: 2
- Unknown: 3

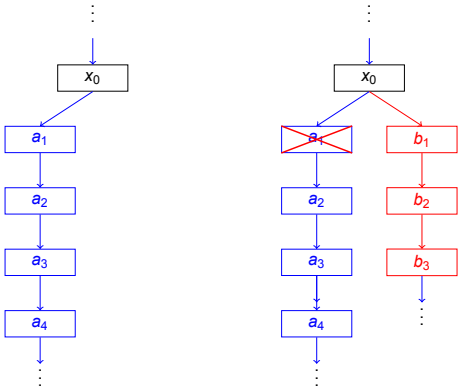
Extrapolating the Bitcoin CVE list

- Netsplit
  - Unintentional soft or hard fork
  - Consensus level inconsistencies in clients
  - Block/tx hash poisoning
  - Merkle tree issues
- Integer underflow/overflow
  - Transaction amount arithmetic and gas arithmetic
- Improper timestamp validation
  - Netsplit + chain wedging + mining difficulty manipulation
- Appsec and network API issues
  - Serialization/deserialization problems, memory corruption, DoS
- Localhost wallet API issues
  - Lack of authentication, CORS header issues, CSRF

Netsplit condition

Two or more active ledgers exist in the network
(for an extended period of time)

- block x_0 validated by all nodes
- then comes block a_1
  - some nodes validate it fine (nodes *A*)
  - ...but nodes *B* reject it
- nodes *B* fork off to their own branch

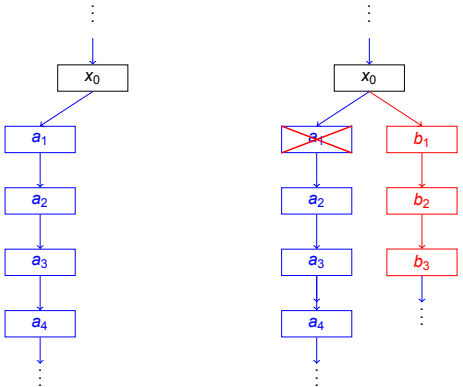


Netsplit condition

Is the split going to self-heal?

- nodes *B* can't switch to *A*'s chain
- nodes *A* *could* switch to *B*
- which chain has more hashing power

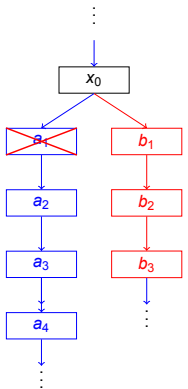
If branch *A* has more hashing power, the split is permanent



Netsplit facilitates the double-spend attack

General idea:

- Execute a tx on chain *A*
- Wait for confirmation, exchange for goods
- Chain *A* will be dropped, tx reverted
- Presumably the same tx does not exist on chain *B*?



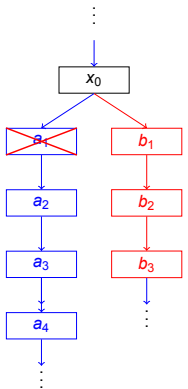
Netsplit facilitates the double-spend attack

In reality, it's not that simple:

- Nodes on chain *B* will also see the tx
  - If broadcasted, the tx will reach nodes *B*
- Some miners offer private tx mining
- Tx is still public and can be replayed on chain *B*

An attack variant:

- 1. Mine the tx privately on chain *A*
- 2. Once it's mined on chain *A*, broadcast the doublespend



Netsplit condition

A closer look at 2 Netsplit Bitcoin CVEs:

- Block hash collision (CVE-2012-2459)
  - Block hash poisoning
- Inconsistent BDB lock limit interactions (CVE-2013-3220)
  - (Unintended) hard fork on software upgrade

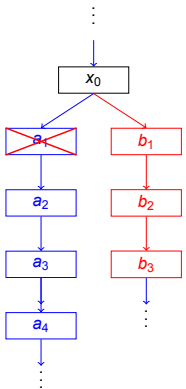
Netsplit via block/tx hash poisoning

Attack goal: get nodes to *reject a valid block*.

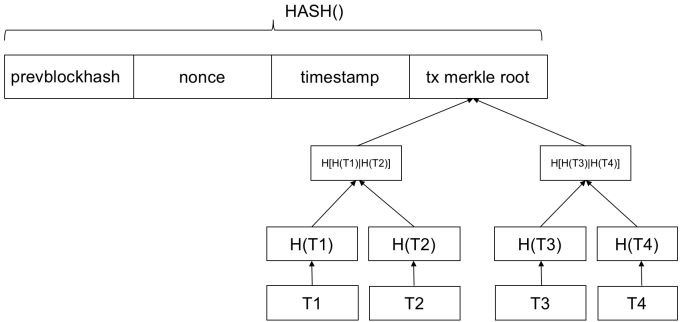
- Block/tx hash poisoning
- Nodes cache blocks' hashes
- Necessary in a P2P setting

Poison a valid block's hash?

- Take note a valid broadcasted block
- Tweak block content to invalidate it
- ...without changing the block's hash
- Is block's hash malleable?



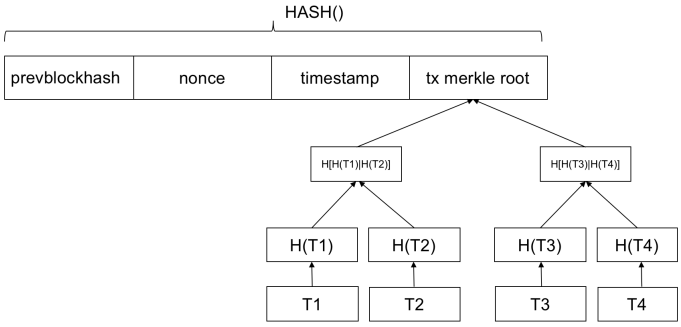
Netsplit via block/tx hash poisoning



What's in a block hash?

- the block header
- ...which includes a hash of the transcations

Netsplit via block/tx hash poisoning

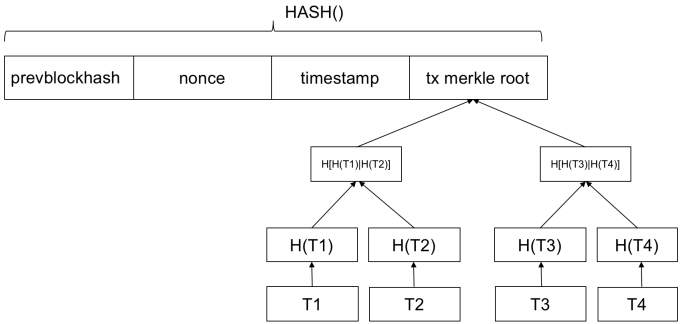


$H(SER(prevblockhash, nonce, timestamp, merkleHash(PAD(T_1, .., T_n))))$

- $SER(\cdot)$ is a serialization method
- $T_1, ..T_n$ is the transaction list
- $PAD(T_1, ..T_n)$: if $n \neq 2^k$

This is different than violating 2nd preimage of the $H(x)$

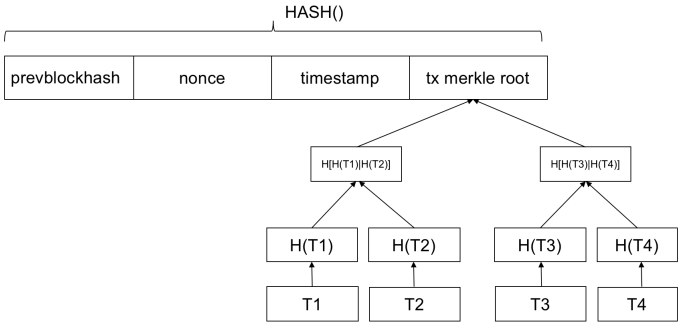
Netsplit via block/tx hash poisoning



...aren’t second pre-image attacks ruled out by the hash function?

- we’re not attacking the hash function
- rather, a construction that relies on it

Netsplit via block/tx hash poisoning

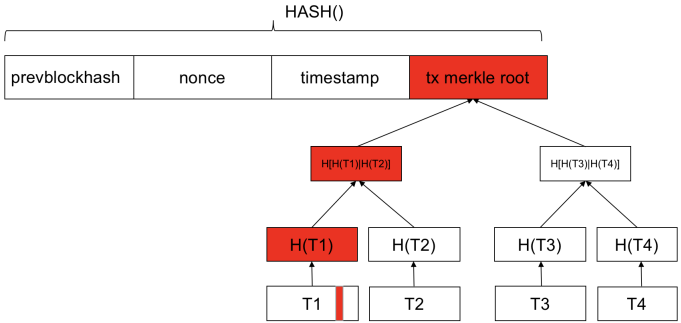


Collision in $SER(\cdot)$? Suppose $SER(\cdot)$ is just a concatenation:

$H(prevBlockHash|nonce|timestamp|merkleHash(PAD(T_1, .., T_n)))$

- Modify the block header:
  - Take last character from ‘nonce‘
  - Prepend it to ‘timestamp‘
  - Different block header serializes to the same byte string
  - Collision in $SER(\cdot)$

Netsplit via block/tx hash poisoning

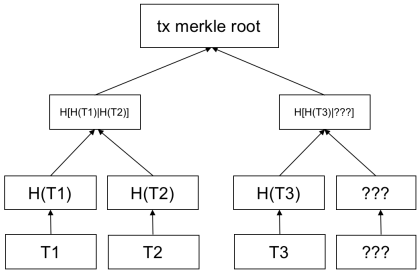


$H(SER(prevblockhash, nonce, timestamp, merkleHash(PAD(T_1, .., T_n))))$

Try to find collision by attacking the Merkle hash calculation?

- if the number of txs is 2^m
- no hope for collision

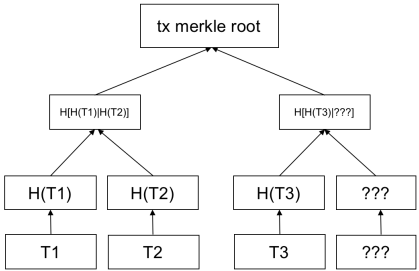
Netsplit via block/tx hash poisoning



What if the number of txs is not 2^k?

- Padding: $merkleRoot(pad(T_1, \ldots T_n))$
  - $pad(T_1, T_2, T_3) = (T_1, T_2, T_3, T_3)$
  - $pad(T_1, T_2, T_3, T_4, T_5) = (T_1, T_2, T_3, T_4, T_5, T_3, T_4, T_5)$
- Trivial collisions on padding
- $pad(T_1, T_2, T_3) = pad(T_1, T_2, T_3, T_3)[= (T_1, T_2, T_3, T_3)]$

Netsplit via block/tx hash poisoning



CVE-2012-2459[1]:

- Observe a block with $n \neq 2^m$ txs
- Add the padded txs as actual block's txs
- Send the block to target nodes
- Validation fails due to duplicate txs
- Correct block gets blacklisted

[1] *forrestv*: Block Merkle calculation exploit: `https://bitcointalk.org/index.php?topic=152282.0`

Netsplit condition

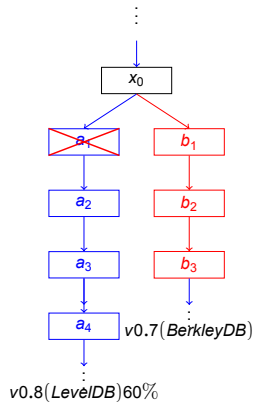
A closer look at 2 Netsplit Bitcoin: CVEs:

- Block hash collision (CVE-2012-2459)
  - Block hash poisoning
- Inconsistent BDB lock limit interactions (CVE-2013-3220)
  - (Unforeseen) hard fork on software upgrade

Berkley DB lock exhaustion hard fork

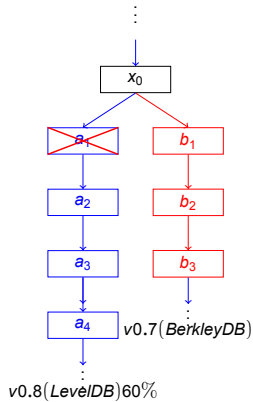
Netsplit during *client software update*:

- A non forking update
- However: unintentional change in the block validity rules
- Replace Berkley DB with LevelDB
- Wikipedia:*BDB can support thousands of simultaneous threads of control or concurrent processes...*
- (new LevelDB) nodes A: v0.8 (60%)
- (old Berkley DB) nodes B: v0.7.x



Berkley DB lock exhaustion hard fork

- BDB config: max. number of simultaneously open locks
- Until that point, processing blocks would not trigger the limit
- Howver, block 225430: unusually high number of tx inputs
- BDB nodes rejected the block
- LevelDB nodes accepted the block



Berkley DB lock exhaustion hard fork

Pages: [1] 2 3 4 5 6 » All print

Author	Topic: A successful DOUBLE SPEND US$10000 against OKPAY this morning. (Read 123464 times)

macbook-air
Sr. Member

Activity: 324
Merit: 250

A successful DOUBLE SPEND US$10000 against OKPAY this morning.
March 12, 2013, 06:22:02 PM #1

All time UTC+08:00:

08:08 – Well before I knew what later have happened, I deposited $10000-worth Bitcoins to BTC-e over OKPAY's Bitcoin payment, I paid OKPAY address 12z2n8YCJw1BEsJhhQPLCTuLqwH341nKnE 211.9093 BTC and 0.0005 BTC as transaction fee.
09:30 – The transaction was included in version 0.8's fork, block 225446
10:08 – Deposit completed, $9800 credited to my BTC-e account
12:53 – After some study, I recognized, the transaction, though included in version 0.8's fork, was never confirmed by the pre-0.8 fork, so I decided to make two double spend transactions on two of the vins of the OKPAY transaction, and broadcasted them with the raw transaction API, 0.001 BTC transaction fee included in each transaction.
13:01 – The double spend transaction was included in pre-0.8 fork block 225446

You should know what happens next...

I bet merchants would think twice before they decide to accept Bitcoins after the incident.

- netsplit condition can be abused for double-spending
- we have two long competing chains without any disbalance in hashing power

The unexplored world of netsplits/forks

- Different client implementations acting on the network:
  - Are we sure they all implement the same protocol?
  - Result: security = full equivalence of the implementations
- Execution environment discrepancies:
  - Architectural differences (OS, 32 vs. 64-bit, architecture, etc)
  - Language undefined behavior?
  - Underlying libraries versions fixed?
- Client software upgrades:
  - Any change in consensus-critical code?
  - Any underlying library upgraded?
  - May go unnoticed

Bitcoin CVEs

Page Discussion Read View source View history Search Bitcoin Wiki

Common Vulnerabilities and Exposures

CVE	Announced	Affects	Severity	Attack is...	Flaw	Net
Pre-BIP protocol changes	n/a	All Bitcoin clients	Netsplit[1]	Implicit[2]	Various hardforks and softforks	100%
CVE-2010-5137	2010-07-28	wxBitcoin and bitcoind	DoS[3]	Easy	OP_LSHIFT crash	100%
CVE-2010-5141	2010-07-28	wxBitcoin and bitcoind	Theft[4]	Easy	OP_RETURN could be used to spend any output.	100%
CVE-2010-5138	2010-07-29	wxBitcoin and bitcoind	DoS[3]	Easy	Unlimited SigOp DoS	100%
CVE-2010-5139	2010-08-15	wxBitcoin and bitcoind	Inflation[5]	Easy	Combined output overflow	100%

Around 30 CVEs

- DoS: 14
- Netsplit: 4
- Theft: 4
- Exposure: 4
- Inflation: 2
- Unknown: 3

A sample of Bitcoin DoS CVEs

Just as any other attack surface DoS concerns:

- Crashes:
  - Assert failure (e.g. CVE-2018-17144)
  - Divide by zero in Bloom filter handling CVE-2013-5700
- CPU exhaustion:
  - Unlimited SigOp DoS (CVE-2010-5138)
  - Multiple DoS vectors in orphan transaction handling (CVE-2012-3789)[2]
- Memory/space exhaustion
  - Memory exhaustion with excess tx message data (CVE-2013-4627)
- Network exhaustion
  - Nodes exchange huge amount of data over the network (CVE-2013-4627)

[2] Sergio Demian Lerner `https://en.bitcoin.it/wiki/CVE-2012-3789`

Application level DoS and blockchain

Compare DoS in blockchain with DoS in other software

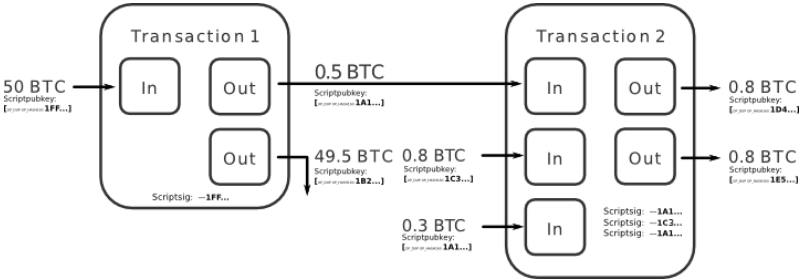
- Web-app DoS? Affects availability
  - ...does *not* affect other security goals (such as *confidentiality*)
- DoS in blockchain: affects security properties beyond availability
  - ...DoS may result in reduced network's hash power
  - Less security against double spend
- DoS vectors have higher severity in blockchain
- ...and should be taken seriously

CPU and memory exhaustion bugs

As in any other application:

- Arbitrary or large amount of CPU computation?
  - $O(n^2)$ algorithms
  - Iterating over sets with user controlled size
  - Cryptographic operations
- Unlimited memory or disk stores? For instance:
  - Uncofirmed tx pool (mempool)
  - Orphan transaction pool

Orphan tx handling DoS



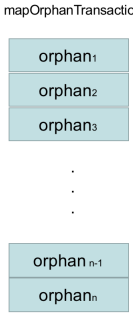
Bitcoin txs:

- Can have multiple inputs and outputs
- Input: hash and output index
- All referenced inputs are spent in full

What's an orphan tx?

- One or more inputs (parents) txs are unknown
- Why keep them? The tx order may change

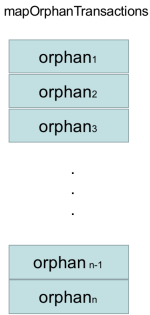
Orphan tx handling DoS



How are orphan txs stored?

```
map<uint256, CDataStream*> mapOrphanTransactions;
```

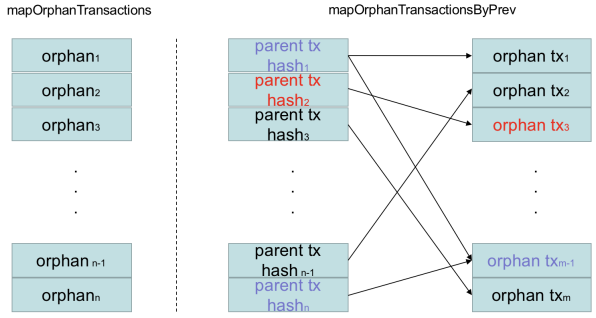
Orphan tx handling DoS



What happens when a regular (non-orphan) tx is ingested?

- Some of the orphans need to be *unorphaned*
- Given a new tx, which ones should be *unorphaned*?

Orphan tx handling DoS




```
multimap<uint256, CDataStream*> mapOrphanTransactionsByPrev;
```

New tx arrives:

- Just index `mapOrphanTransactionsByPrev`
- ...and know what to unorphan

Orphan tx handling DoS

Lack of size limit on the orphan memory size:

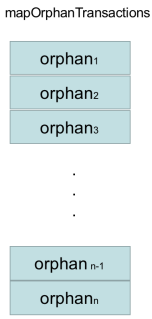
- DoS by exhausting client's memory size
- Limit introduced: 10k orphans max
- ...limit on the size of each orphan

Ejection policy?

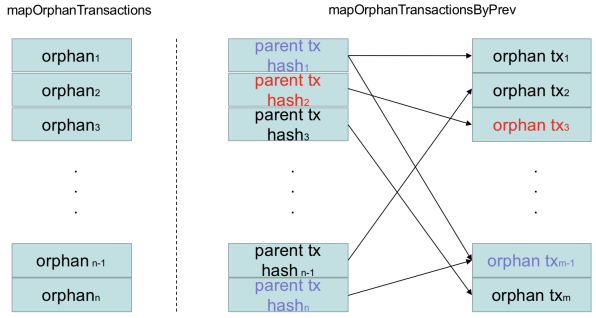
- Randomly pick an orphan and delete it

Memory exhaustion resolved. However...

- How does orphan deletion work exactly?



Orphan tx handling DoS



How to delete an orphan tx?

- Delete *all edges pointing to the target tx*
- Used to look up the orphan tx's parent tx hashes
- Example: `orphan tx_m-1`

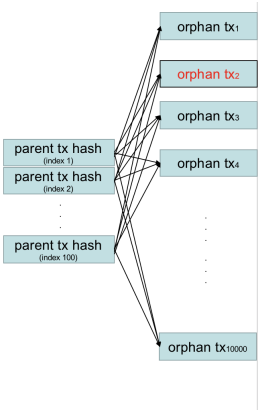
Orphan tx handling DoS

- Orphan tx deletion: iterates over branches
- Some branches need to be deleted, some not
- The number of branches is attacker-controlled
- Such a pattern can be turned into a DoS
- The iteration count becomes is large
- Each requests is computationally demand
- Send many requests

Orphan tx handling DoS

CPU exhaustion CVE-2012-3789

- Have the target store 10k orphans
- ...and a huge number of branches
- One parent tx, different outputs
- Send another orphan: trigger *ejection*
- Deletion now triggers
  - Filtering a 1 million branch set



Auditing crypto currency software

Pick you target and start by looking for:

- Netsplit fork
  - Unintentional soft or hard fork [1,2]
  - Client state inconsistencies [3,4]
  - Merkle tree issues [5]
- Integer underflow/overflow
  - Transaction amount arithmetic and gas arithmetic [6]
- Improper timestamp validation
  - Netsplit + chain wedging + mining difficulty manipulation [7]
- Appsec and network API issues
  - Serialization/deserialization problems, memory corruption, DoS [8]
- Localhost wallet API issues
  - Lack of authentication, CORS header issues, CSRF [9]

References

[1]: https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki and https://bitcointalk.org/index.php?topic=152282.0;all

[2]: https://bitcoin.stackexchange.com/questions/54878/why-is-it-so-hard-for-alt-clients-to-implement-bitcoin-core-consensus-rules

[3]: https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures#CVE-2012-1909

[4]: https://bitcoin.stackexchange.com/questions/5903/where-can-i-learn-more-about-bip30-namely-the-exploit-and-the-background-discus/5905#5905, https://github.com/bitcoin/bips/blob/master/bip-0030.mediawiki

[5]: https://bitcointalk.org/?topic=102395, https://bitslog.com/2018/06/09/leaf-node-weakness-in-bitcoin-merkle-tree-design/

[6]: https://en.bitcoin.it/wiki/Value_overflow_incident

[7]: http://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html

[8]: https://bitslog.com/2013/07/18/buggy-cve-2013-4627-patch-open-new-vectors-of-attack/, https://bitcoin.stackexchange.com/questions/83485/serialized-transaction-bigger-than-the-actual-transaction-object-cve-2013-4627 + vario

[9]: https://medium.com/@lukedashjr/cve-2018-20587-advisory-and-full-disclosure-a3105551e78b

Depressing the Crypto Economy with DoS Bugs

Aleks Kircanski

CackalackyCon, June, 2019

nccgroup