Incentivizing consensus research before forks

The recent consensus issue after the Berlin fork has highlighted the need to spend more resources on ensuring there's no way to break consensus between different clients. I suggest a decentralized bounty that incentivizes security researchers to find such corner cases and report them on a testnet rather than mainnet.

The high level idea

Make a minimal modification to all clients, to let users get a proof of a state change and the transactions that led to it. Deploy a bounty contract to mainnet, which pays the bounty to the first researcher who proves breaking the state on a testnet.

The required change

Each client (e.g. geth, OpenEthereum) should implement a new RPC call, eth_stateCommitment(blocknum)

which works for recent blocks and returns a state-change commitment signed by the node's private key:

{ 'blockNum':num, 'preStateRoot':hash, 'postStateRoot':hash, 'transactions':[..], 'signature':{r,s,v} }

The list of transactions is the full list included in that block.

The bounty contract on mainnet

The contract will contain a bounty in ETH, and will have a list of public keys for all testnet nodes included in the game. The list contains nodes from any number of testnets, but never from mainnet nodes.

The contract shall use commit/reveal to ensure that the bounty can be claimed by the researcher rather than a frontrunner. A minimum time of 30 minutes is enforced between commit and reveal.

commit(hash)

- record msg.sender and the hash, which should be keccak(transaction,nonce).

prove(transaction,nonce,signedStateCommitment_1,signedStateCommitment_2)

- prove that consensus was broken in that block. Check if:
- keccak(transaction,nonce) was committed by msg.sender.
- Both state commitments have the same block number, the same preStateRoot, and the same list of transactions.
- ecrecover both signatures, ensure that they differ and that both are registered testnet nodes on the same testnet.
- The two postStateRoots differ.

If all are true, send the entire balance to msg.sender. An event is emitted to alert the client maintainers.

At this point the bounty contract is empty so no further claims will be paid. The bounty will be replenished only after the bug is fixed and all nodes are upgraded. This ensures that each bug is only paid once, and only to the researcher who found it.

The flow of an "attack"

Alice researched two different clients and found a way to break consensus. E.g. a transaction involving a new precompile. She follows the following procedure:

1. Prepare the transaction (or sequence of transactions) that will trigger the bug
2. Prepare the commitment. If the attack involves multiple transactions, she only prepares a commitment for the final transaction in the sequence, the one that triggers the bug. Alice calculates keccak(transaction,random_nonce) and sends it to commit().
3. Ensure that the commitment has been recorded under her own address. If Alice has been frontrun, she repeats the process with a new nonce.
4. Perform the attack on a testnet which participates in the bounty.
5. Contact two testnet nodes of different types, and requests eth_stateCommitment(blocknum) for the block that included the transaction.
6. Verify that the two state commitments have different post states, indicating that the attack has succeeded.

7. Call prove(transaction,nonce,signedStateCommitment_1,signedStateCommitment_2)
8. Profit!

At that point the client maintainers are alerted, given the transaction, and can research and fix the bug.

Caveat

The bounty could also be claimed by an attacker who manages to steal the private key of one of the testnet nodes, by signing a bad state commitment. While this doesn't reveal a consensus bug, it does indicate that a specific client has been hacked and is therefore worth knowing as well.