Preliminary

: The Blockchain at Berkeley Plasma team is currently involved in implementing the plasma-mvp spec using the Cosmos sdk for the sidechain development. www.github.com/fourthstate. We've arisen at a potential attack vector on the rootchain contract with the current spec that would allow an attacker to repeatedly exit a UTXO.

Problem

: An adversary deposits amount X, and creates a chain of transactions on the sidechain to addresses that the adversary owns, each transaction of amount X is included in different blocks. The adversary never broadcasts any of the confirm signatures to the network (no need, they can create it for themselves), and can therefore exit each utxo without the possibility of a successful challenge on the rootchain. This will drain the root contract of its funds.

Example:

Let Malicious Actor Eve owns addresses A, B, and C. Eve then makes transactions:

A -> B -> C

Where each transfer is for 1 Eth.

All these transactions get included in their own blocks. Eve does not broadcast the confirm signatures because she owns the addresses of A, B, and C. Eve withdraws A and succeeds. She then creates the confirm signature for transaction A -> B and withdraws B successfully. She then creates confirm signature for transaction B -> C and withdraws C. Eve can withdraw A, B, then C without challenge due to the lack of a broadcasted confirm signature. Through this attack, Eve drains the smart contract, potentially regaining the gas cost from the second withdrawal. In this example, B itself can represent several more transactions between A and C.

In the previous attack, if B is exited anytime during or before the challenge period of A, the confirm signature of A sending to B can be extracted. As an alternative, the adversary can use all the transactions between A and C as filler and only exit transaction utxo A and C. None of the confirm signatures of these filler transactions are ever broadcasted which prevents a successful challenge.

_

The attack vector above is not possible on Plasma Cash due to the non-fungibility of tokens.

_

Naive Solution

:

If Eve tries to exit A and C, as in the scheme above, a naive solution would be to allow any client to challenge the exit of C with a proof that 1.) A has been exited 2.) C is a "descendent" of A. This can be achieved through a series of merkle proofs that show all transactions between A and C have been spent.

However, this is not practical because of the non-existent upper bound on the filler transactions between A and C. The gas cost for challenging a transaction can be made arbitrarily large by adding more filler transactions in between exits. Thus, the attacker can make the cost of challenging greater than any reward given to a successful challenger.

Potential Solution

:

Add a requirement that all valid transactions include the confirm signatures for the inputs being spent. For a given UTXO, anyone can now recover all confirm signatures (except the signature of the most recent ancestor) by traversing through the transaction history of the UTXO. Since confirm signatures must be included in a transaction, any malicious individual sending transactions to themselves will be forced to include the signatures in the transaction, and therefore, anyone can challenge an invalid withdrawal. Finding the correct confirm signatures to challenge correctly is linear in the number of blocks since the exit.

For example, if Eve owns the addresses A, B, C, and sends transactions from A -> B, and B -> C, and tries to exit A, anyone can challenge this exit by providing the confirm signature included in transaction B -> C.

An edge case we found: If Eve sends a transaction to herself, represented as A -> B, then the confirm signature for this will not be included in the block unless she spends B.

Case 1: Eve withdraws A before withdrawing B: in this case, we add an additional challenge that shows 1.) A is being withdrawn or already withdrawn 2.) B is a direct descendent of A.

Case 2: Eve withdraws B before withdrawing A: in this case, we add an event in the root contract that broadcasts confirm signatures used to withdraw B that can be used to invalidate the withdrawal of A.

Any feedback is appreciated!

- Blockchain at Berkeley, Plasma Team