Getting funds into the escrow in sequential commit

Note: content of this section assumes that priceDiscoveryContract is of a simple type (denoted "Fulfil Orderbook" in BPIP4)

ASK flow:
In sequential commit, reseller does not have a “seller pool” to fund, so the dynamic is slightly different than for an initial commit. Reseller just needs to approve the protocol to transfer the exchange token at the end of the exchange (to fill up the escrow).

1. ***Can the buyer (who is executing the tx) set a higher price in the “*PriceDiscovery*” struct than it was agreed to? There are 3 reasons why this could happen:***
    a. ***buyer tries to trick the protocol to pull more reseller’s funds into escrow, maybe with the intent to get excessive buyer protection or simply to mess up with the reseller***
    b. ***Price on “*priceDiscoveryContract*” actually changed. This could for example happen if “*priceDiscoveryContract*” implements a bonding curve and pool properties changed right before sequential commit started***
    c. ***By mistake***

    First thing to notice is that the buyer must always pay the price set in “*PriceDiscovery*” otherwise our protocol reverts immediately.
    Protocol then forwards all funds to “*priceDiscoveryContract*”, which should compare received funds with funds determined by the “*priceDiscoveryData*”. It’s up to “*priceDiscoveryContract*” to properly handle the surplus of received funds. In general, we can expect one of the following behaviours:
    a. “*priceDiscoveryContract*” reverts and consequently, the exchange is not executed. Protocol reverts and all funds are returned.
    b. “*priceDiscoveryContract*” returns the surplus to msg.sender (in this case protocol contract). In this case, the protocol knows what the actual price is (userSuppliedPrice - returnedFunds) and adjusts all calculations (immediate payout, escrow amount) to that price. Surplus is then returned to the buyer.
    c. “*priceDiscoveryContract*” forwards the surplus to the reseller. In this case, we can consider userSuppliedPrice as the true price. And since the buyer actually paid for it, it’s ok if the protocol transfers the full minimal escrow amount from the reseller into the protocol.
    d. “*priceDiscoveryContract*” simply keeps the surplus to itself. In this case, the protocol still considers userSuppliedPrice as the true price and takes the full amount from the reseller.

    Cases a, b and c pose no threat to either buyer or reseller. Either everything is returned (case a), a true agreed price is taken into account (b) or buyer buyer-supplied price is accepted (c). The last option can be seen as the buyer actually paying more than necessary, but not more than they are willing to pay and also getting full protection for what they paid.
    Option d is the problematic one and protocol cannot distinguish between c and d solely on the response it gets from “*priceDiscoveryContract*”.

- For a buyer it's the least problematic. In case the exchange finalises happily, they just paid more than they needed to. If a dispute is raised, they actually have full protection and can get everything back.
- For a reseller it poses an active threat since the protocol locks up funds based on userSuppliedPrice and not on agreed price in "*priceDiscoveryData*". If a dispute is raised they can lose more than they received on secondary trade.

However, this still does not break. Because this is an "ask" flow, the reseller had to approve "*priceDiscoveryContract*" to transfer the voucher. So when they give this approval, they know what the behaviour of "*priceDiscoveryContract*" is. If they know they can lose money they simply won't choose this kind of discovery mechanism.
So the final thing is to clarify what happens if the caller supplies "*priceDiscoveryContract"* that was not approved to transfer the reseller's voucher:

1. Expected behaviour of (honest) "*priceDiscoveryContract*" is to simply revert and in this case protocol will also revert. So no funds are lost on either side.
2. But if "*priceDiscoveryContract*" is malicious (presumably constructed by a malicious buyer) it might return gracefully and try to trick the protocol that the order was fulfilled successfully. But this attempt still doesn't work. One of the steps is that the protocol must transfer the voucher to the buyer. But if the voucher was never transferred (because *priceDiscoveryContract* was not approved to do it), protocol does not own the voucher and therefore cannot execute this step and the whole transaction is reverted.
   Note: it is important that the protocol actually verifies that it *owns* the voucher, not that it just tries to transfer it. The transfer could succeed even if it does not own it, in case the reseller approves protocol to transfer it. This might happen for different reasons (by mistake, was tricked into it, or they simply started to execute BID order). If the protocol does not check actual ownership, a malicious buyer could exploit it.

2. ***Can the buyer (who is executing the tx) set a lower price in the "*PriceDiscovery*" struct than it was agreed to? There are 2 reasons why this could happen:***
   a. ***They think they can get away with a lower price and protocol will cover for the deficit***
   b. ***By mistake***

   Protocol will always validate that the buyer sent the amount they specified in the "*PriceDiscovery*" struct and will forward the full amount to the "*priceDiscoveryContract"*. There are two expected behaviours:
   1. Honest "*priceDiscoveryContract*" will revert and protocol will revert.
   2. Malicious (or badly implemented) "*priceDiscoveryContract*" will return gracefully.

   Similarly as above, resellers should never approve a bad "*priceDiscoveryContract*", which means the protocol will never receive the voucher and the whole transaction will be reverted.

BID flow:
Bid flow is less complex, since the reseller only approves protocol to transfer the voucher. Buyer does not approve the boson protocol, but "*priceDiscoveryContract".*

3. ***Can the reseller (who is executing the tx) set a lower price in the "*PriceDiscovery*" struct than it was agreed to? There are 3 reasons why this could happen:***
    a. ***seller tries to trick the protocol to pull less reseller funds into escrow,***
    b. ***Price on "*priceDiscoveryContract*" actually changed. This could for example happen if "*priceDiscoveryContract*" implements a bonding curve and pool properties changed right before sequential commit started***
    c. ***By mistake***

    In bid flow protocol first transfers the reseller's voucher to itself and then it acts as the seller in the exchange, meaning that protocol will receive all proceeds when a call to "*priceDiscoveryContract*" is made, assuming "*priceDiscoveryContract*" is honest and well implemented. Protocol can then compare received funds to userSuppliedPrice.
    1. If "*priceDiscoveryContract*" returns more than or equal to expected, this actually received funds are considered the full price, and immediate payout and escrow amount are calculated based on that price. By definition escrow amount cannot be higher than secondary price, so protocol already has more than needed and it just pays out the reseller the difference between secondary price and escrow amount.
    2. If "*priceDiscoveryContract*" returns less than price, protocol should just revert. If not, malicious "*priceDiscoveryContract*" could try to get a voucher for a lower price than expected.

    Another thing to consider is, what happens if "*priceDiscoveryContract*" actually sends less to the seller than it collects from the buyer. There are various reason this can happen:
    a. "*priceDiscoveryContract*" collects some service fee
    b. "*priceDiscoveryContract*" is not performing exchanges atomically, i.e. it just collects seller's proceeds and allow them to withdraw them separately
    c. "*priceDiscoveryContract*" is a malicious contract (presumably created by the seller) to collect full price from the buyer, and give them false sense of buyer protection, while not really ensuring against original seller default

    In (a), service fee doesn't get insured, so the buyer can potentially lose it in case of the original seller's default. But since this is the bid flow, the buyer is the one that can choose "*priceDiscoveryContract"* and can decide for some without the service fee. Or they can choose it and bear the risk.
    Both options (b) and (c) can also lead to loss of buyer's funds, so they should not even approve "*priceDiscoveryContract*" to transfer the funds in the first place. In (c) this is actually a problem outside the protocol, because if the buyer approves it, the seller doesn't even need a protocol to steal the funds. And case (b) is actually bad for both parties - if "*priceDiscoveryContract*" puts the proceeds into escrow, the reseller could get them at all, since they would belong to boson protocol contract. And because buyer can also lose funds (if original seller defaults), there is no incentive for anyone to use this kind of "*priceDiscoveryContract*"

4. ***Can the reseller (who is executing the tx) set a higher price in the "*PriceDiscovery*" struct than it was agreed to? There are 2 reasons why this could happen:***

a. ***seller tries to get higher price out of the protocol than agreed with the buyer***
b. ***By mistake***

As described above, in “bid” flow protocol acts as an intermediary seller and knows exactly how much it receives after “*priceDiscoveryContract*' fulfils the order. If the received value is less than userSuppliedPrice it reverts.
If it ever happened that “*priceDiscoveryContract*” actually returned more than agreed price was that is a problem outside the of protocol:

a. If it has an ability to somehow steal the buyers funds, we cannot prevent it. But in that case the attacker doesn’t even need our protocol.
b. If it sends its own (i.e. contract’s) funds, that’s again a flaw in the contract itself and to exploit it, one doesn’t need a protocol.
c. If it would be specifically tailored to work with our protocol (for example a reseller tries to manipulate escrow), the attacker would first need to fund the contract themselves, and at the end the protocol would lock even more than the buyer paid in. But it all goes at the expense of the attacker, so there is no incentive to do it.