BZx Exploit 1 - Summary

- Not a direct oracle exploit
- Logic bug in code
- Fairly sophiscated

Step 1

- We need liquidity!
- Flash loan from dYdX
- 0x1E0447b19BB6EcFdAe1e4AE1694b0C3659614e4e.operate
- Balance: 10,000ETH

Step 2

- Use ETH to borrow WBTC (cTokens)

  Substep 1: 0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5.mint() - To mint ETH equivalent of cTokens

  5500ETH sent and 5500cETH minted.

  5500cETH sent to cWBTC, to borrow wBTC.

  Substep 2: 0xC11b1268C1A384e55C48c2391d8d480264A3A7F4.borrow() - To borrow cWBTC equivalent of 5500cETH - Total 112WBTC.

- Balance: 4500ETH, 112WBTC.

Step 3 – Big Brain Time pt1

- 1300ETH /WBTC short position on BZx

  0xb0200B0677dD825bb32B93d055eBb9dc3521db9D.mintWithEther

- 1300ETH is converted into WBTC collateral of 5x short. This has to happen:

  Substep 1: Borrow x4 from iETH pool (so total x5).

  Substep 2: Convert all ETH (iETH) to WBTC - Looks up Kyber for liquidity (only Uniswap has sufficiently). Price of WBTC to ETH spikes up.

- 5x short, i.e. 20% collateral buffer. So if slippage <20%, all is good. Total WBTC purchased more than borrowed amount.

But here... Slippage is >20% due to a big volume buy on a market with thin liquidity.

So why did the slippage protection not kick in?

Step 3 – Big Brain Time pt2

- In short:
  0xb0200b0677dd825bb32b93d055ebb9dc3521db9d.mintWithEther(2 params) calls mintWithEther(3 params) without loanDataBytes, and cascades ALL the way down... Subsequently doesn't trigger.

```
require ((
            loanDataBytes.length == 0 && // Kyber only
            sentAmounts[6] == sentAmounts[1]) || // newLoanAmount
        !OracleInterface(oracle).shouldLiquidate(
            loanOrder,
            loanPosition
        ),
        "unhealthy position"
    );
```

- Balance: 3200ETH, 112WBTC. 51WBTC in BZx. And a super cheap WBTC to ETH rate on Uniswap (0.0163)

Step 4

- Sell 112 WBTC back into Uniswap
- 112WBTC/0.0163 = 6871ETH

- Balance: 10,071ETH. 51 in BZx.

Step 5

- Repay dYdX as per Step 1
- Left with ~65ETH and 5500ETH position in Compound (Step 2)
- Repay 112WBTC for 5500ETH.
- Assuming 1BTC = 36ETH, spend 112*36=4032ETH, to get 5500ETH.

BZx Exploit 2

- An oracle/arbitrage trade
- Legitimate market manipulation
- Simple contract, straightforward, and super effective.

Step 1

- We need liquidity!
- Flash loan from BZx lol
- 0x77f973fcaf871459aa58cd81881ce453759281bc.flashBorrowToken
- Balance: 7500ETH

Step 2

- Spend 900ETH to buy sUSD from Kyber
- Increases sUSD price
- 1*540ETH tx, 18*20ETH txs.
- Balance: 6600ETH

Step 3

- Exchange ETH to sUSD
- 0x172e09691dfbbc035e37c73b62095caa16ee2388.exchangeEtherForSynths

  3,517.859101713642764819 Ether (3517 got exchanged)

  2,482.14089828635723518 Ether (refunded)

- Balance: 600ETH leftover + 2482.14ETH refunded = 3082.14ETH

Step 4

- Back to BZx contract lol

  0x85CA13d8496b2D22D6518fAeB524911E096Dd7E0.borrowTokenFromDeposit
- Remember sUSD is priced >1USD, and BZx uses Kyber price feed.
- For 3517.85 ETH worth of sUSD, get 6796ETH.

- Balance: 600+2482.14+6796 = 9878.14ETH

Step 5

- Payback BZx for 7500ETH.
- Balance 9878.14 – 7500 = 2378.14ETH.

How would the exploit contract look like?

0x77f973fcaf871459aa58cd81881ce453759281bc.flashBorrowToken

0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2.withdraw

0x818e6fecd516ecc3849daf6845e3ec868087b755.swapEtherToToken

0x172e09691dfbbc035e37c73b62095caa16ee2388.exchangeEtherForSynths

0x77f973fcaf871459aa58cd81881ce453759281bc.borrowTokenFromDeposit (using synth as collateral to get ETH)

At this point the attacker doesn't even need to repay the stuff done on step 5, and walk away with the profit after repaying flash loan.

PS: Need to double check contract addresses – Whether pointing to proxy or implementation.