Bonq DAO oracle manipulation attack

Team 5

Agenda

1. Background
2. Attack
3. Post-mortem
4. Actions-Taken
5. Tutorial
6. Mitigations & best practices

Bonq DAO

Bonq is a non-custodial, decentralized and over-collateralized lending platform that solves four critical problems for projects and protocols that have a token:

- Allows them to borrow against their own tokens at zero interest rate
- Creates deep liquidity solution without the need to incentivize or pay the other side of the liquidity pool
- Offers sustainable yields to their community members holding tokens in a safe and secure environment
- Allows treasuries to de-risk and create a smart capital allocation

How does it work?

Users can access the liquidity of their own digital assets by locking them up in a trove, which is a smart contract controlled only by the users, and mint a low volatility payment coin BEUR, pegged to the Euro.

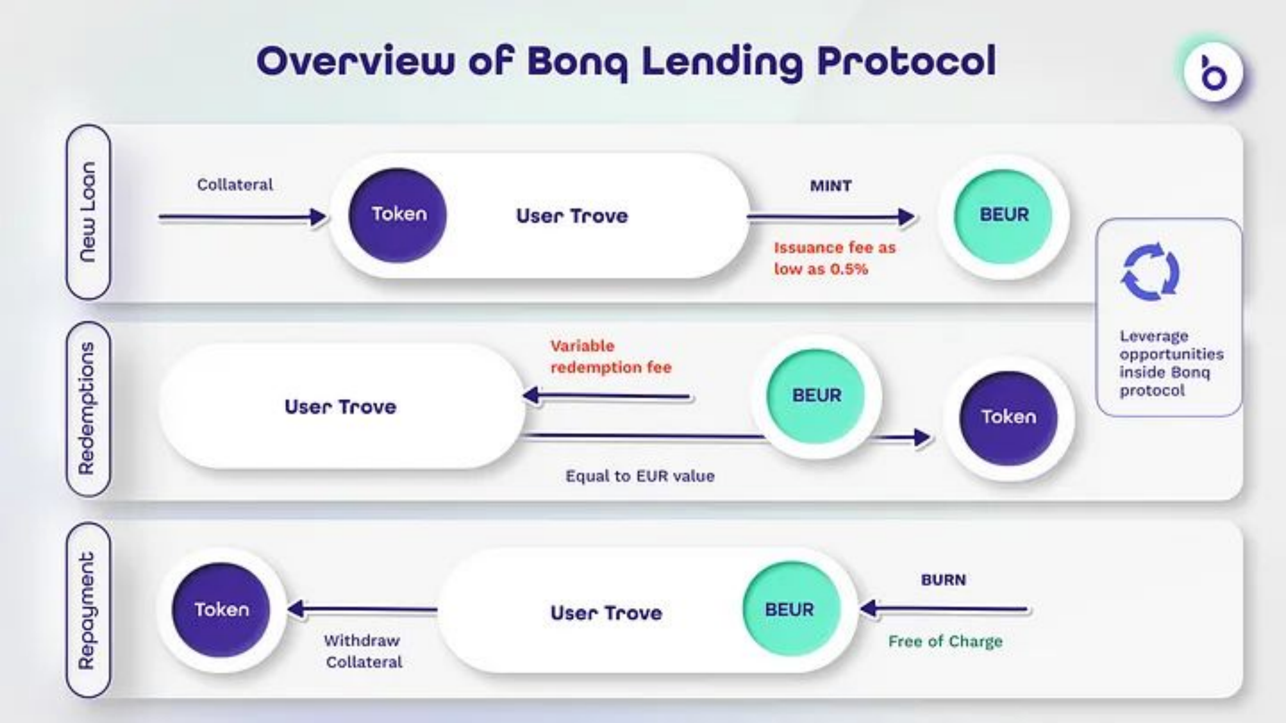
Fees: minting and redeeming but not repaying.

Risks: Trove Liquidation and market volatility.

Properties of BEUR

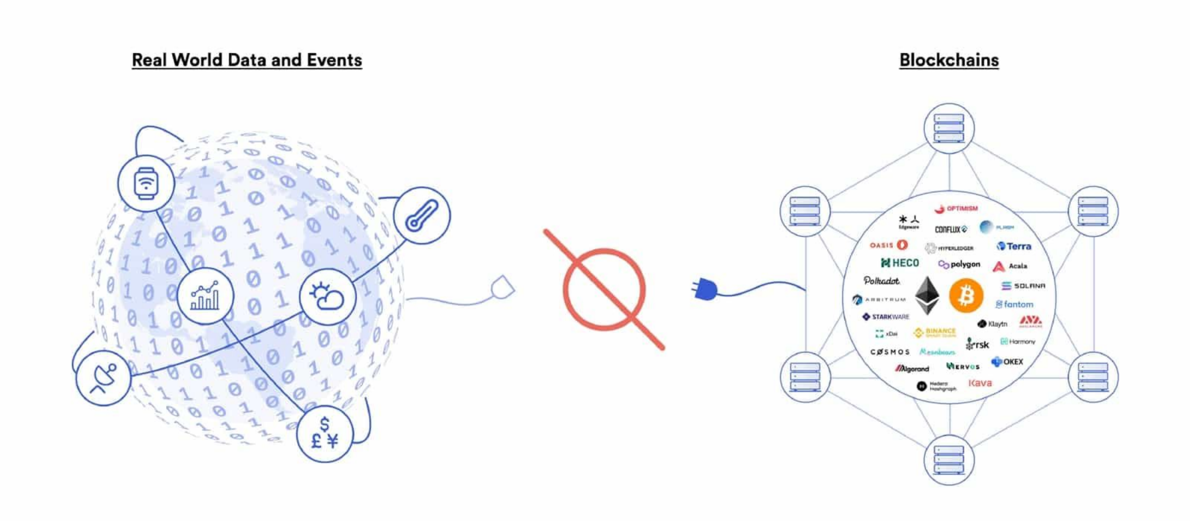
- Tradable and exchangeable - Anyone can buy or sell them, whether they have an open Trove or not
- Fully redeemable - Users can always swap 1 BEUR for 1 EUR worth of collateral (minus fees) directly on the Bonq platform. This will create a floor for BEUR price
- Mintable - Users mint BEUR in their trove smart contracts
- Burnable - BEUR tokens are burned when used to repay a Trove’s debt

Bonq



Credit: https://blog.allianceblock.io/abfundrs-getting-to-know-bonq-f470b4e4bb3b

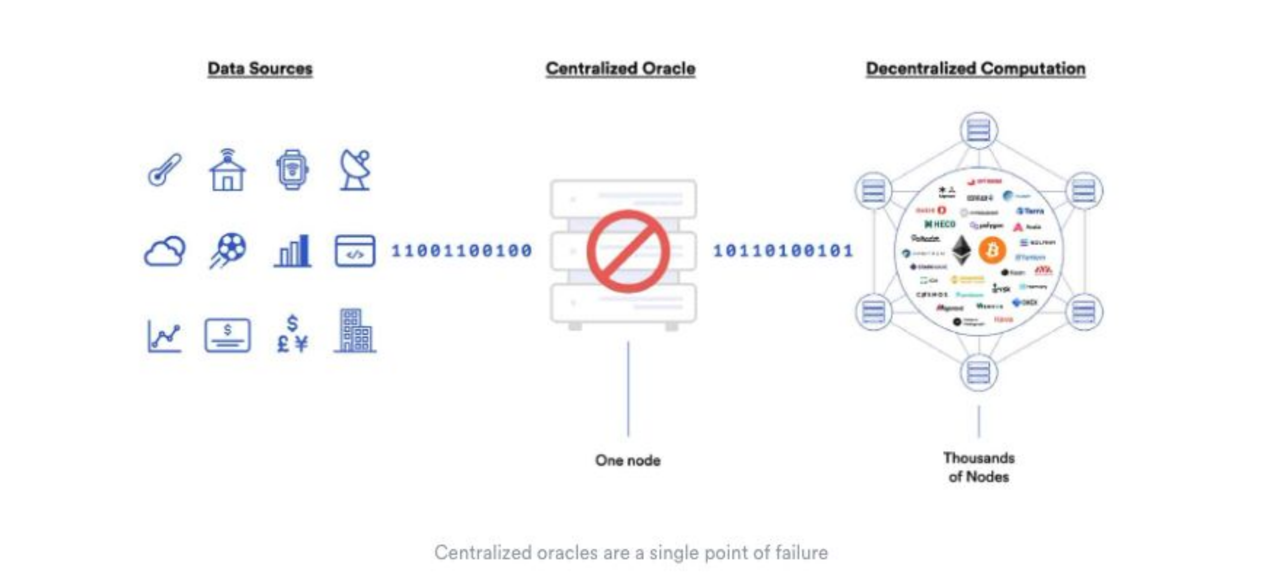
Oracles in Ethereum ecosystem: Chainlink



Credit: chainlink.com

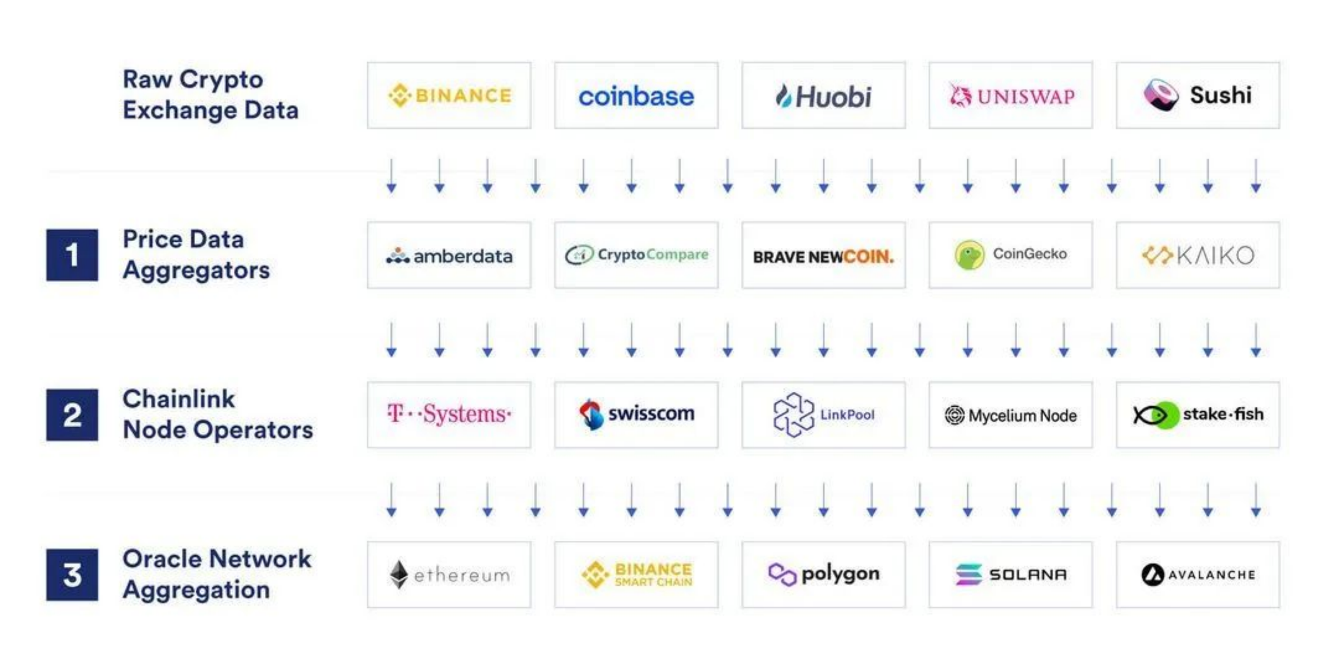
Oracles provide a way for the decentralized Web3 ecosystem to access existing data sources, legacy systems, and advanced computations

Oracles: Centralized V/s Decentralized



Credit: chainlink.com

Oracles: Centralized V/s Decentralized



Raw Crypto Exchange Data

1 Price Data Aggregators

2 Chainlink Node Operators

3 Oracle Network Aggregation

Credit: chainlink.com

The Tellor system

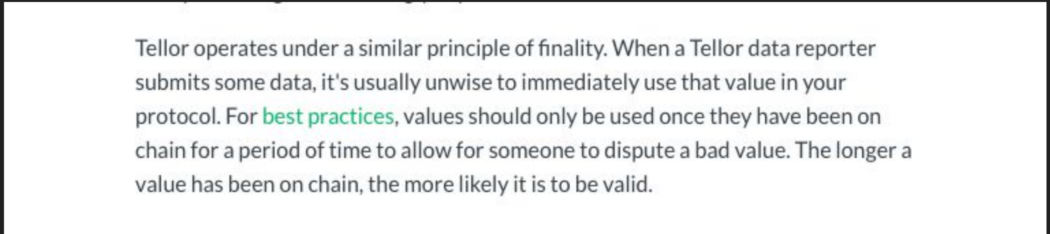
Tellor solves the problem by aligning the incentives of data reporters, data consumers, and Tellor token holders.

In brief, anyone can deposit a stake and report data. For a period of time, anyone can pay a dispute fee to challenge any piece of data.

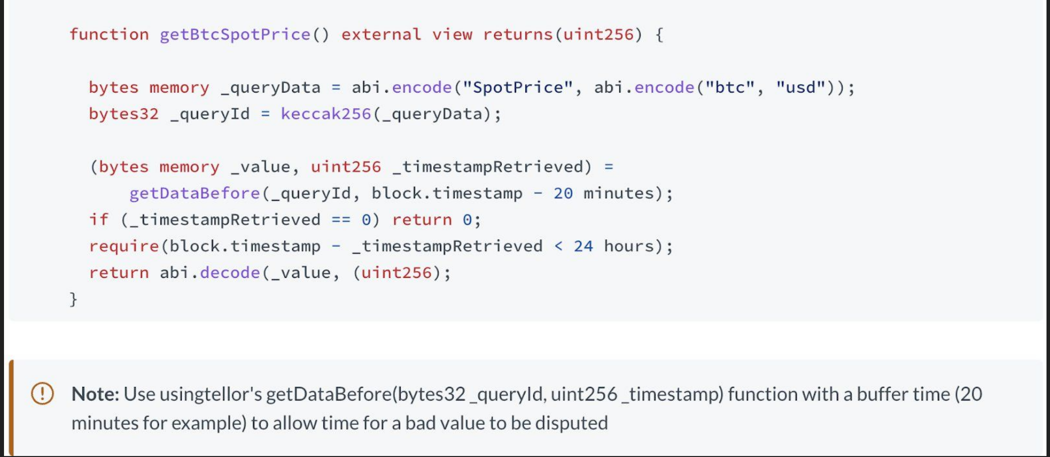
Tellor stakeholders vote to determine the outcome of the dispute.

If the data reporter loses the dispute, the reporter's stake goes to the disputing party. This creates a system where bad actors are punished and good actors are rewarded.

Tellor System: Caveats



Tellor operates under a similar principle of finality. When a Tellor data reporter submits some data, it's usually unwise to immediately use that value in your protocol. For best practices, values should only be used once they have been on chain for a period of time to allow for someone to dispute a bad value. The longer a value has been on chain, the more likely it is to be valid.



```
function getBtcSpotPrice() external view returns(uint256) {

  bytes memory _queryData = abi.encode("SpotPrice", abi.encode("btc", "usd"));
  bytes32 _queryId = keccak256(_queryData);

  (bytes memory _value, uint256 _timestampRetrieved) =
      getDataBefore(_queryId, block.timestamp - 20 minutes);
  if (_timestampRetrieved == 0) return 0;
  require(block.timestamp - _timestampRetrieved < 24 hours);
  return abi.decode(_value, (uint256));
}
```

Note: Use usingtellor's getDataBefore(bytes32 _queryId, uint256 _timestamp) function with a buffer time (20 minutes for example) to allow time for a bad value to be disputed

Credit: tellor.io

The exploit

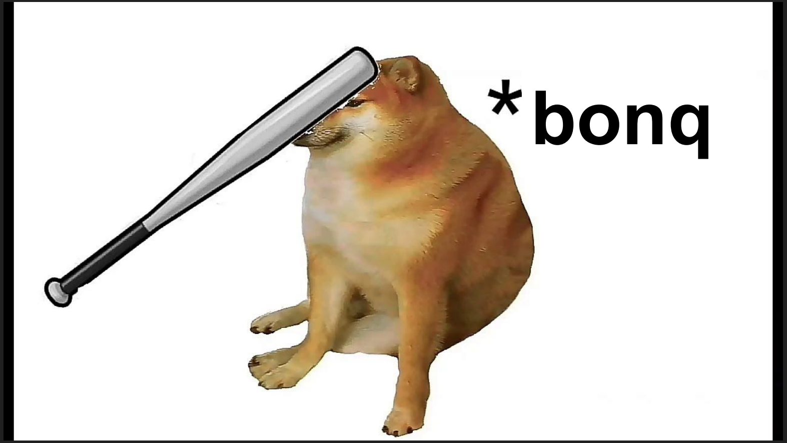
The Attack

1. Bonq Dao uses Tellor as an oracle.
2. Tokens involved in attack:

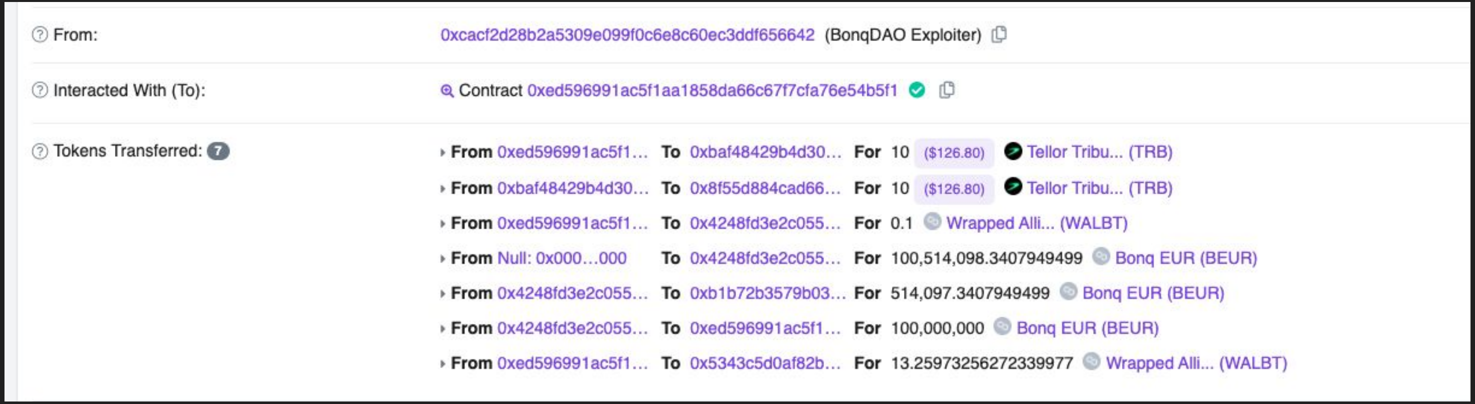
BEUR

ALBT

Why ALBT? The most liquid token on Bonq



The Attack



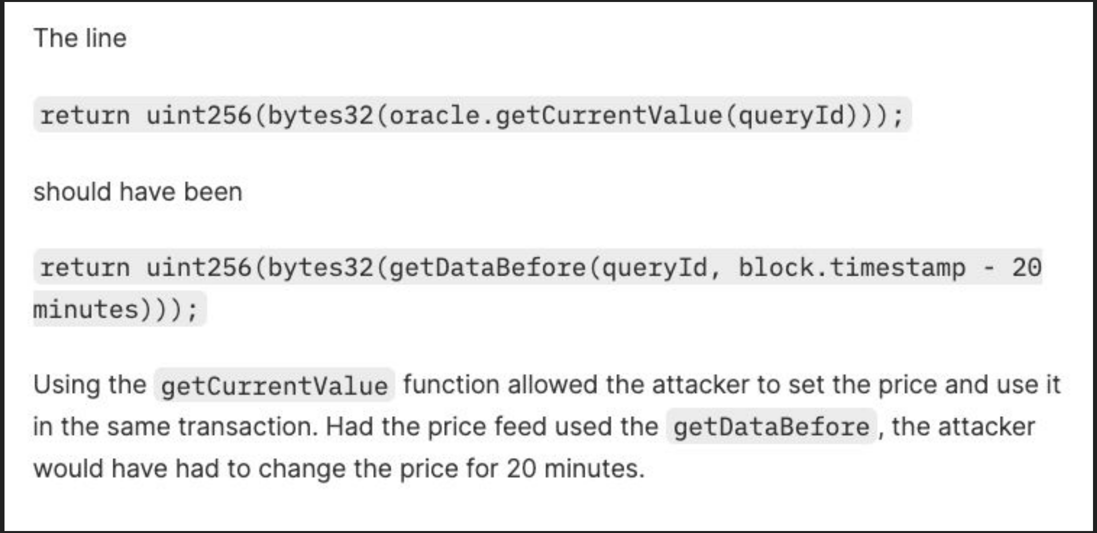
From: 0xcacf2d28b2a5309e099f0c6e8c60ec3ddf656642 (BonqDAO Exploiter)

Interacted With (To): Contract 0xed596991ac5f1aa1858da66c67f7cfa76e54b5f1 ✓

Tokens Transferred: 7

- From 0xed596991ac5f1… To 0xbaf48429b4d30… For 10 ($126.80) Tellor Tribu... (TRB)
- From 0xbaf48429b4d30… To 0x8f55d884cad66… For 10 ($126.80) Tellor Tribu... (TRB)
- From 0xed596991ac5f1… To 0x4248fd3e2c055… For 0.1 Wrapped Alli... (WALBT)
- From Null: 0x000…000 To 0x4248fd3e2c055… For 100,514,098.3407949499 Bonq EUR (BEUR)
- From 0x4248fd3e2c055… To 0xb1b72b3579b03… For 514,097.3407949499 Bonq EUR (BEUR)
- From 0x4248fd3e2c055… To 0xed596991ac5f1… For 100,000,000 Bonq EUR (BEUR)
- From 0xed596991ac5f1… To 0x5343c5d0af82b… For 13.25973256272339977 Wrapped Alli... (WALBT)

Post-mortem



The line

```
return uint256(bytes32(oracle.getCurrentValue(queryId)));
```

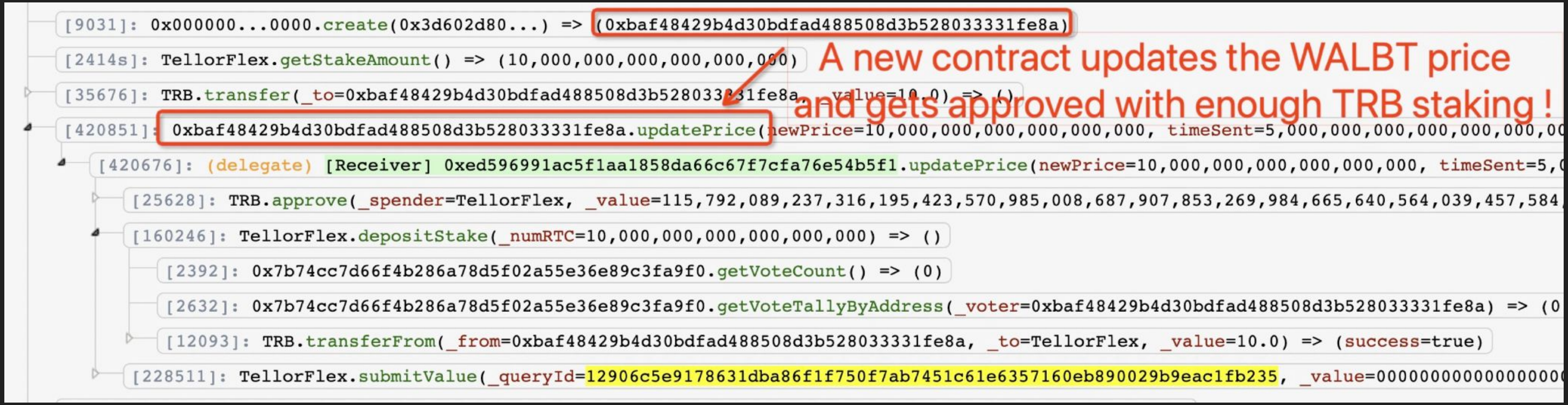
should have been

```
return uint256(bytes32(getDataBefore(queryId, block.timestamp - 20
minutes)));
```

Using the `getCurrentValue` function allowed the attacker to set the price and use it in the same transaction. Had the price feed used the `getDataBefore`, the attacker would have had to change the price for 20 minutes.

Source: BonQ DAO analysis

Post-mortem

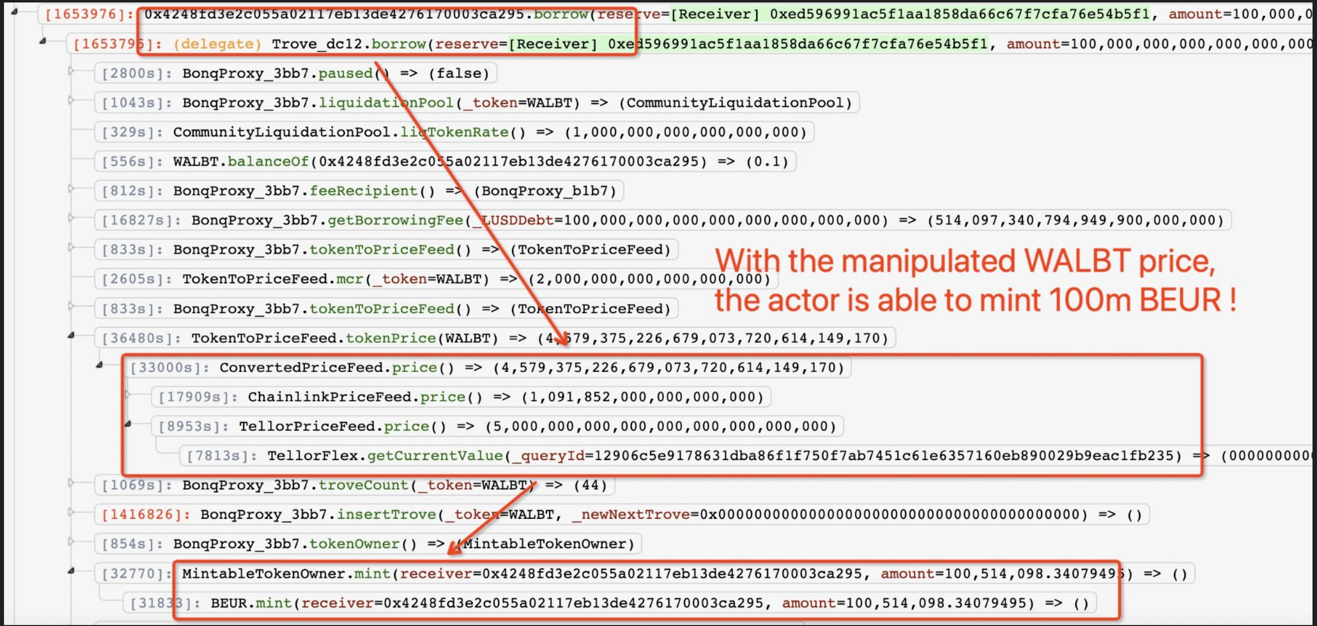


```
[9031]: 0x000000...0000.create(0x3d602d80...) => (0xbaf48429b4d30bdfad488508d3b528033331fe8a)
[2414s]: TellorFlex.getStakeAmount() => (10,000,000,000,000,000,000)
[35676]: TRB.transfer(_to=0xbaf48429b4d30bdfad488508d3b528033331fe8a, _value=10.0) => ()
[420851]: 0xbaf48429b4d30bdfad488508d3b528033331fe8a.updatePrice(newPrice=10,000,000,000,000,000,000, timeSent=5,000,000,000,000,000,000,00
    [420676]: (delegate) [Receiver] 0xed596991ac5f1aa1858da66c67f7cfa76e54b5f1.updatePrice(newPrice=10,000,000,000,000,000,000, timeSent=5,(
        [25628]: TRB.approve(_spender=TellorFlex, _value=115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584
        [160246]: TellorFlex.depositStake(_numRTC=10,000,000,000,000,000,000) => ()
            [2392]: 0x7b74cc7d66f4b286a78d5f02a55e36e89c3fa9f0.getVoteCount() => (0)
            [2632]: 0x7b74cc7d66f4b286a78d5f02a55e36e89c3fa9f0.getVoteTallyByAddress(_voter=0xbaf48429b4d30bdfad488508d3b528033331fe8a) => (0
            [12093]: TRB.transferFrom(_from=0xbaf48429b4d30bdfad488508d3b528033331fe8a, _to=TellorFlex, _value=10.0) => (success=true)
        [228511]: TellorFlex.submitValue(_queryId=12906c5e9178631dba86f1f750f7ab7451c61e6357160eb890029b9eac1fb235, _value=0000000000000000
```

A new contract updates the WALBT price and gets approved with enough TRB staking !

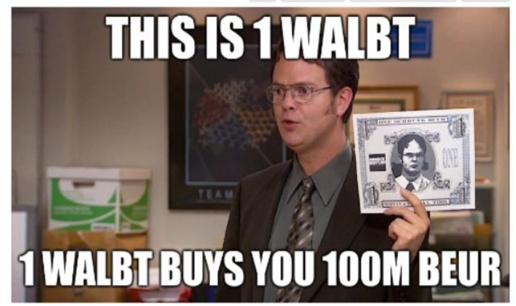
Credit: Peckshield

Post-mortem



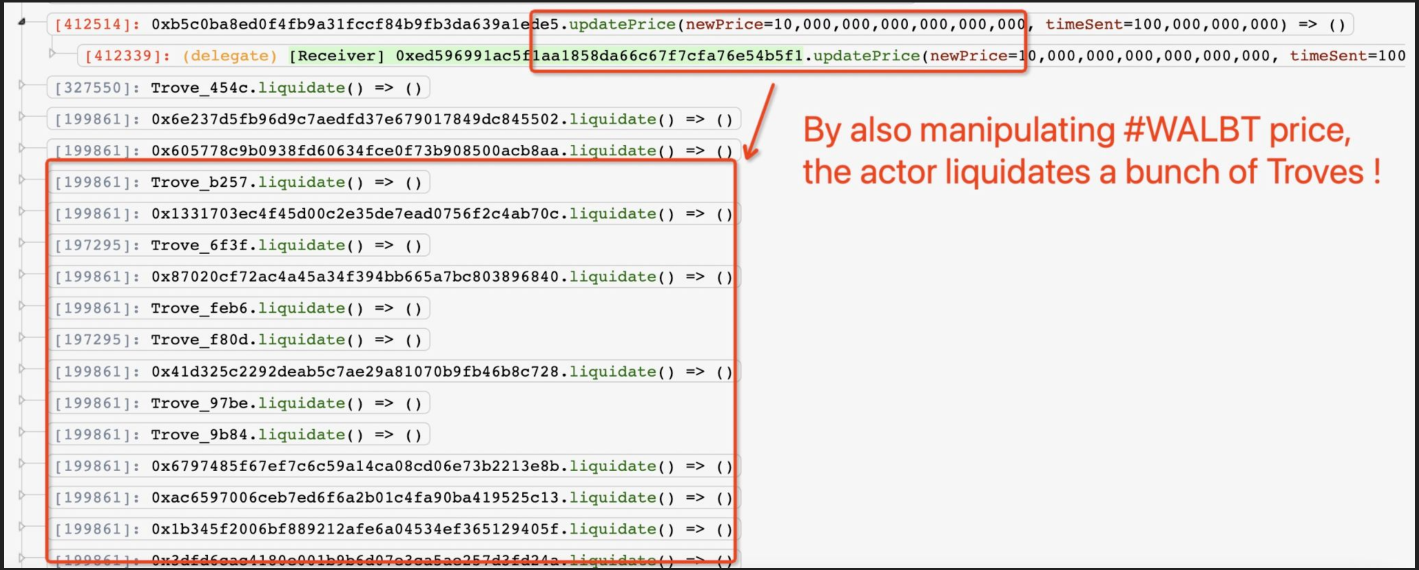
```
[1653976]: 0x4248fd3e2c055a02117eb13de4276170003ca295.borrow(reserve=[Receiver] 0xed596991ac5f1aa1858da66c67f7cfa76e54b5f1, amount=100,000,0
  [1653795]: (delegate) Trove_dc12.borrow(reserve=[Receiver] 0xed596991ac5f1aa1858da66c67f7cfa76e54b5f1, amount=100,000,000,000,000,000,000
    [2800s]: BonqProxy_3bb7.paused() => (false)
    [1043s]: BonqProxy_3bb7.liquidationPool(_token=WALBT) => (CommunityLiquidationPool)
    [329s]: CommunityLiquidationPool.liqTokenRate() => (1,000,000,000,000,000,000)
    [556s]: WALBT.balanceOf(0x4248fd3e2c055a02117eb13de4276170003ca295) => (0.1)
    [812s]: BonqProxy_3bb7.feeRecipient() => (BonqProxy_b1b7)
    [16827s]: BonqProxy_3bb7.getBorrowingFee(_LUSDDebt=100,000,000,000,000,000,000,000,000) => (514,097,340,794,949,900,000,000)
    [833s]: BonqProxy_3bb7.tokenToPriceFeed() => (TokenToPriceFeed)
    [2605s]: TokenToPriceFeed.mcr(_token=WALBT) => (2,000,000,000,000,000,000)
    [833s]: BonqProxy_3bb7.tokenToPriceFeed() => (TokenToPriceFeed)
    [36480s]: TokenToPriceFeed.tokenPrice(WALBT) => (4,579,375,226,679,073,720,614,149,170)
      [33000s]: ConvertedPriceFeed.price() => (4,579,375,226,679,073,720,614,149,170)
        [17909s]: ChainlinkPriceFeed.price() => (1,091,852,000,000,000,000)
        [8953s]: TellorPriceFeed.price() => (5,000,000,000,000,000,000,000,000,000)
          [7813s]: TellorFlex.getCurrentValue(_queryId=12906c5e9178631dba86f1f750f7ab7451c61e6357160eb890029b9eac1fb235) => (000000000
    [1069s]: BonqProxy_3bb7.troveCount(_token=WALBT) => (44)
    [1416826]: BonqProxy_3bb7.insertTrove(_token=WALBT, _newNextTrove=0x0000000000000000000000000000000000000000) => ()
    [854s]: BonqProxy_3bb7.tokenOwner() => (MintableTokenOwner)
    [32770]: MintableTokenOwner.mint(receiver=0x4248fd3e2c055a02117eb13de4276170003ca295, amount=100,514,098.34079495) => ()
      [31833]: BEUR.mint(receiver=0x4248fd3e2c055a02117eb13de4276170003ca295, amount=100,514,098.34079495) => ()
```

With the manipulated WALBT price,
the actor is able to mint 100m BEUR !



Credit: Peckshield

Post-mortem

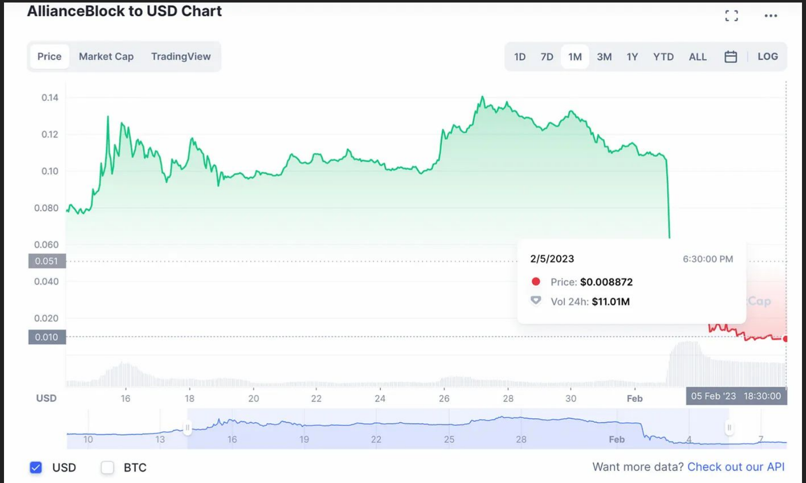


```
[412514]: 0xb5c0ba8ed0f4fb9a31fccf84b9fb3da639a1ede5.updatePrice(newPrice=10,000,000,000,000,000,000, timeSent=100,000,000,000) => ()
    [412339]: (delegate) [Receiver] 0xed596991ac5f1aa1858da66c67f7cfa76e54b5f1.updatePrice(newPrice=10,000,000,000,000,000,000, timeSent=100
[327550]: Trove_454c.liquidate() => ()
[199861]: 0x6e237d5fb96d9c7aedfd37e679017849dc845502.liquidate() => ()
[199861]: 0x605778c9b0938fd60634fce0f73b908500acb8aa.liquidate() => ()
[199861]: Trove_b257.liquidate() => ()
[199861]: 0x1331703ec4f45d00c2e35de7ead0756f2c4ab70c.liquidate() => ()
[197295]: Trove_6f3f.liquidate() => ()
[199861]: 0x87020cf72ac4a45a34f394bb665a7bc803896840.liquidate() => ()
[199861]: Trove_feb6.liquidate() => ()
[197295]: Trove_f80d.liquidate() => ()
[199861]: 0x41d325c2292deab5c7ae29a81070b9fb46b8c728.liquidate() => ()
[199861]: Trove_97be.liquidate() => ()
[199861]: Trove_9b84.liquidate() => ()
[199861]: 0x6797485f67ef7c6c59a14ca08cd06e73b2213e8b.liquidate() => ()
[199861]: 0xac6597006ceb7ed6f6a2b01c4fa90ba419525c13.liquidate() => ()
[199861]: 0x1b345f2006bf889212afe6a04534ef365129405f.liquidate() => ()
[199861]: 0x3dfd6cac4180c001b9b6d07e3ca5ac257d3fd24a.liquidate() => ()
```

By also manipulating #WALBT price, the actor liquidates a bunch of Troves !

Credit: Peckshield

Impact



AllianceBlock to USD Chart

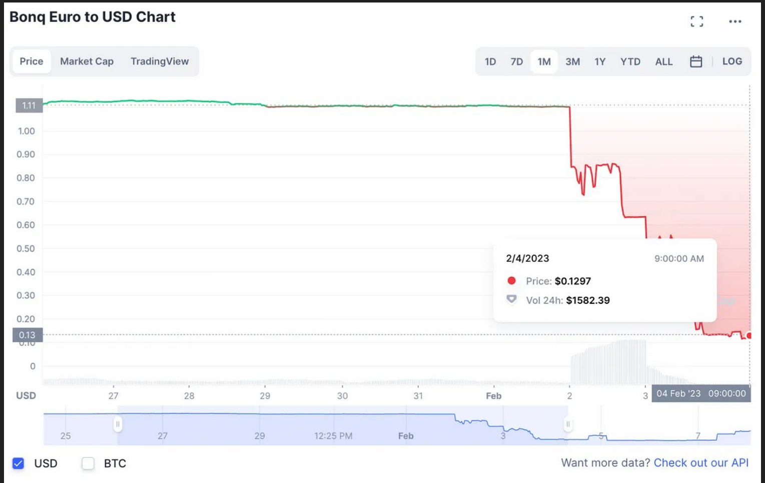
2/5/2023 6:30:00 PM

Price: $0.008872

Vol 24h: $11.01M

05 Feb '23 18:30:00

Want more data? Check out our API



Bonq Euro to USD Chart

2/4/2023 9:00:00 AM

Price: $0.1297

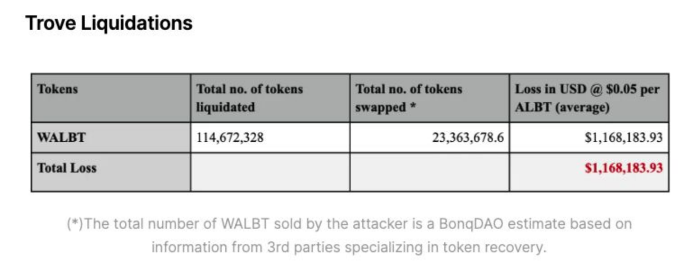
Vol 24h: $1582.39

04 Feb '23 09:00:00

Want more data? Check out our API

Impact

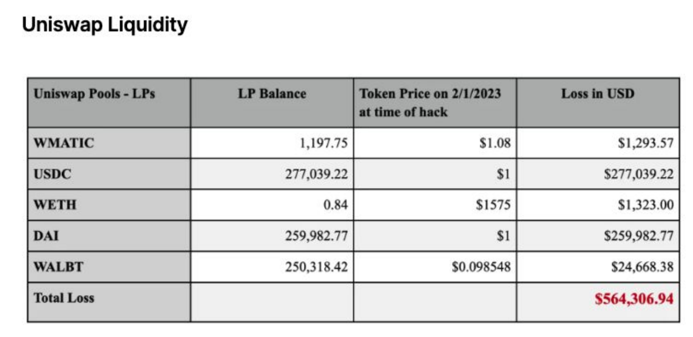
Total damage done: 113.8M #WALBT and 98M #BEUR but not all made it out.



Trove Liquidations

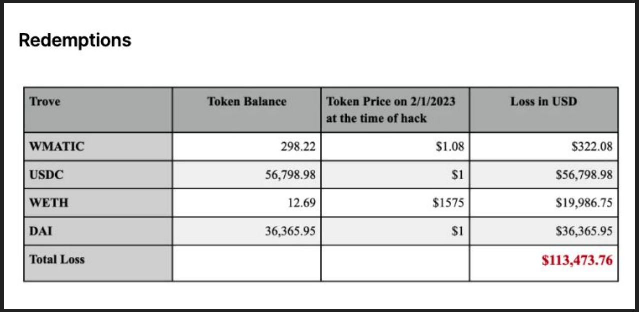
Tokens	Total no. of tokens liquidated	Total no. of tokens swapped *	Loss in USD @ $0.05 per ALBT (average)
WALBT	114,672,328	23,363,678.6	$1,168,183.93
Total Loss			$1,168,183.93

(*)The total number of WALBT sold by the attacker is a BonqDAO estimate based on information from 3rd parties specializing in token recovery.



Uniswap Liquidity

Uniswap Pools - LPs	LP Balance	Token Price on 2/1/2023 at time of hack	Loss in USD
WMATIC	1,197.75	$1.08	$1,293.57
USDC	277,039.22	$1	$277,039.22
WETH	0.84	$1575	$1,323.00
DAI	259,982.77	$1	$259,982.77
WALBT	250,318.42	$0.098548	$24,668.38
Total Loss			$564,306.94

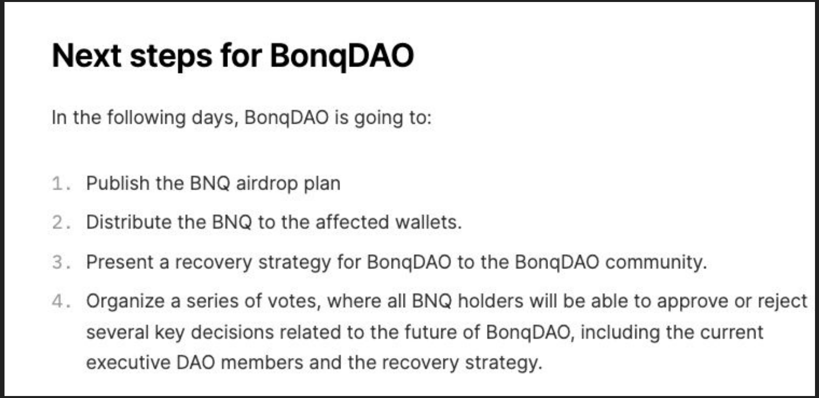


Redemptions

Trove	Token Balance	Token Price on 2/1/2023 at the time of hack	Loss in USD
WMATIC	298.22	$1.08	$322.08
USDC	56,798.98	$1	$56,798.98
WETH	12.69	$1575	$19,986.75
DAI	36,365.95	$1	$36,365.95
Total Loss			$113,473.76

The remaining balance of BEUR in the attacker account has zero value, as there's no more liquidity to trade it.

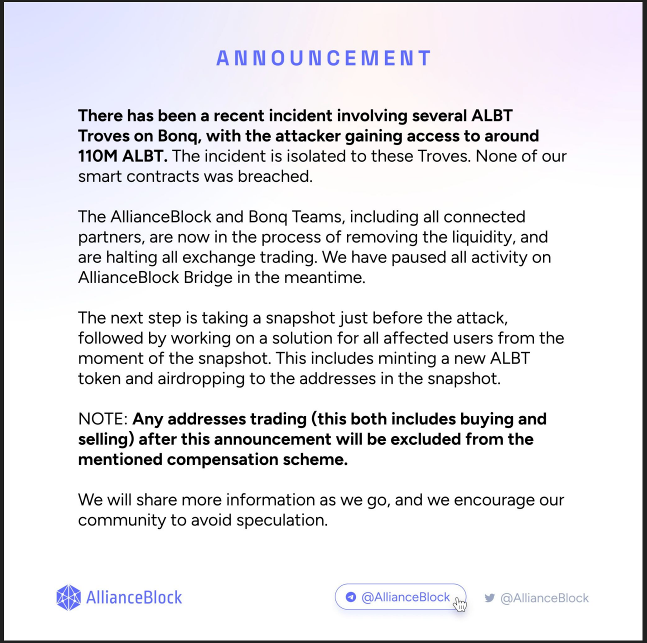
Actions-Taken:



Next steps for BonqDAO

In the following days, BonqDAO is going to:

1. Publish the BNQ airdrop plan
2. Distribute the BNQ to the affected wallets.
3. Present a recovery strategy for BonqDAO to the BonqDAO community.
4. Organize a series of votes, where all BNQ holders will be able to approve or reject several key decisions related to the future of BonqDAO, including the current executive DAO members and the recovery strategy.



ANNOUNCEMENT

There has been a recent incident involving several ALBT Troves on Bonq, with the attacker gaining access to around 110M ALBT. The incident is isolated to these Troves. None of our smart contracts was breached.

The AllianceBlock and Bonq Teams, including all connected partners, are now in the process of removing the liquidity, and are halting all exchange trading. We have paused all activity on AllianceBlock Bridge in the meantime.

The next step is taking a snapshot just before the attack, followed by working on a solution for all affected users from the moment of the snapshot. This includes minting a new ALBT token and airdropping to the addresses in the snapshot.

NOTE: **Any addresses trading (this both includes buying and selling) after this announcement will be excluded from the mentioned compensation scheme.**

We will share more information as we go, and we encourage our community to avoid speculation.

AllianceBlock @AllianceBlock @AllianceBlock

Mitigations and Best Practices

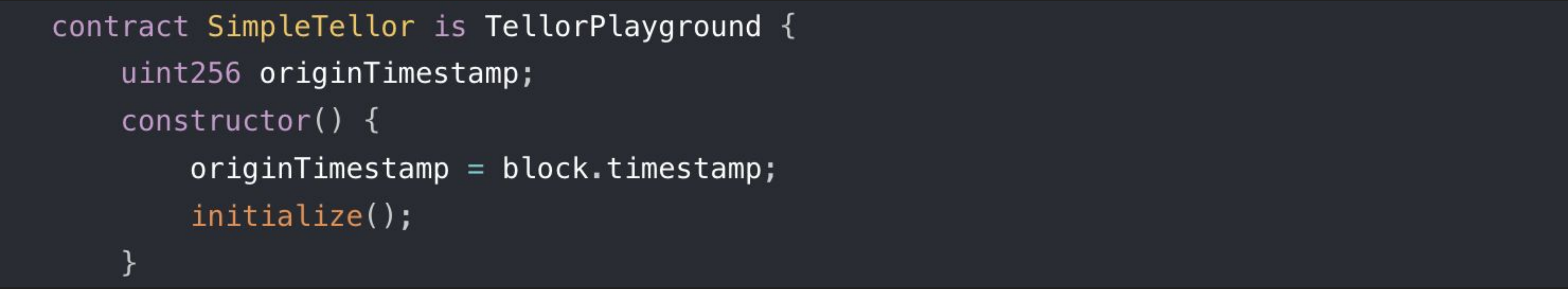
- Use oracles such as Chainlink that provide price update via whitelisted node operators.
- When using a project like Tellor, ensure that the best practices are followed.
- A single person (the CTO) was named as responsible for development and testing - use professional auditing services!
- Large projects should think twice before partnering with lesser known 3rd party projects.

Mitigations and Best Practices

- The protocol using oracle as price feeds must have a time difference in order for the price to be checked and verified before being utilized, and it cannot significantly rely on changes made in the oracle instantly.
- Uniswap V2 introduces a TWAP (Time Weighted Average Price) oracle for use by on-chain developers which is highly resistant to oracle manipulation attacks.
- Protocols need to add security layers, using at least two oracles to verify the price. This would mitigate the hack and ensure proper checks on critical functions and variables that are publicly accessible.

Tutorial: Liquidator

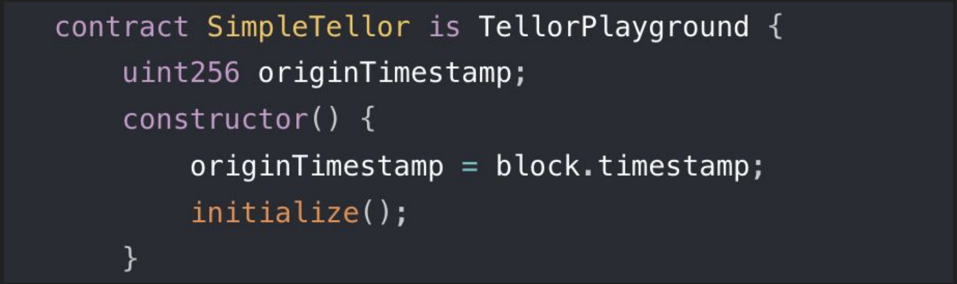
- Actual price of TokenX = 1 TokenUSD
- Manipulate price of TokenX by feeding off-chain price
- Exchange 1 TokenX to 10000 TokenUSD and drain the Liquidator



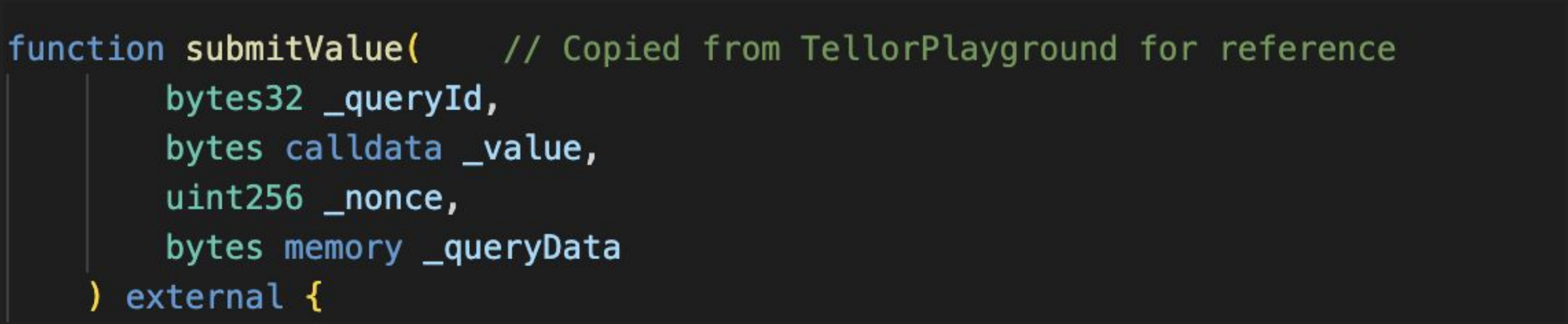
```
contract SimpleTellor is TellorPlayground {
    uint256 originTimestamp;
    constructor() {
        originTimestamp = block.timestamp;
        initialize();
    }
```

Tutorial: Liquidator

- Use the SimpleTellor oracle to manipulate price of TokenX

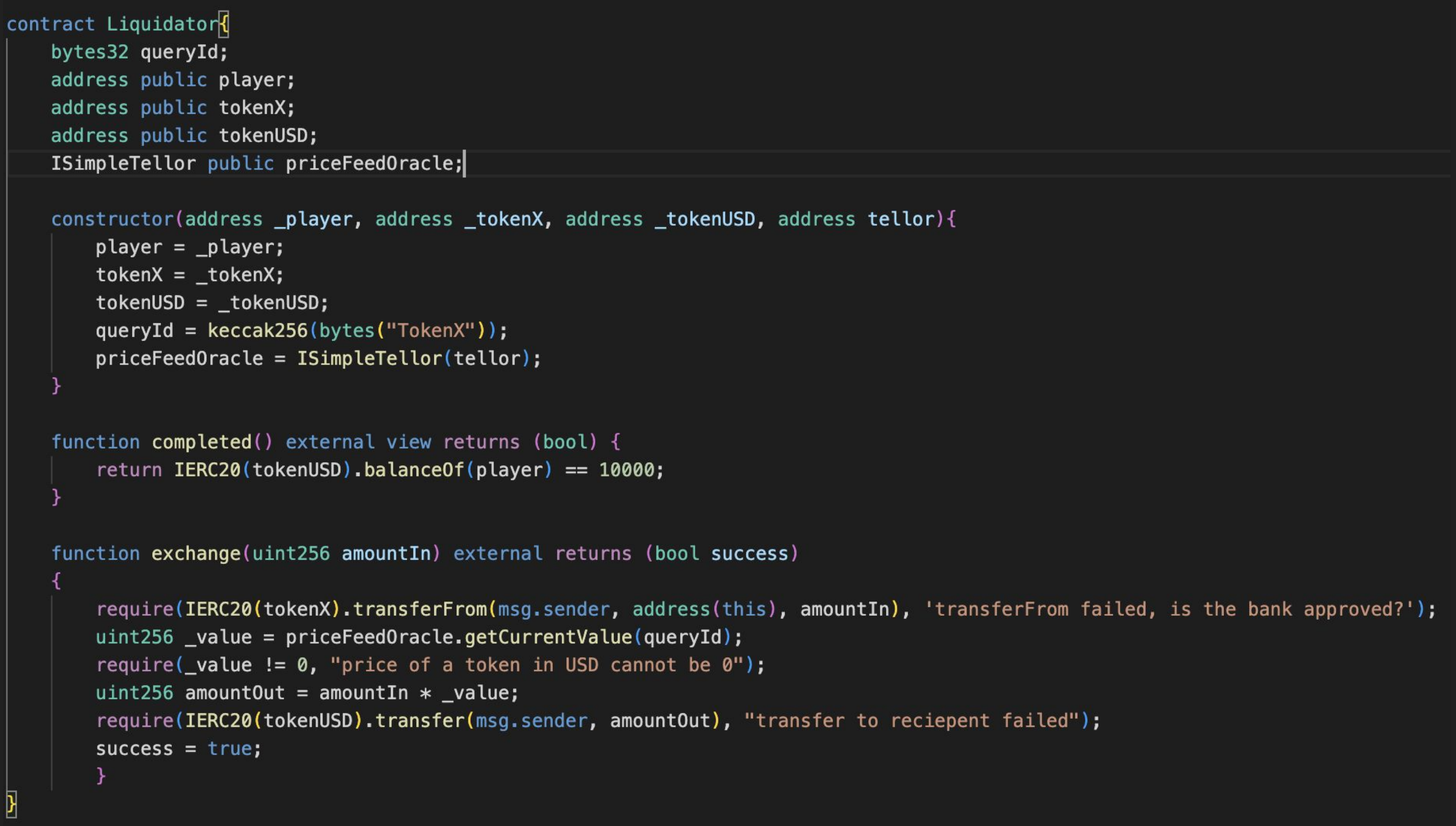


```
contract SimpleTellor is TellorPlayground {
    uint256 originTimestamp;
    constructor() {
        originTimestamp = block.timestamp;
        initialize();
    }
```



```
function submitValue(    // Copied from TellorPlayground for reference
        bytes32 _queryId,
        bytes calldata _value,
        uint256 _nonce,
        bytes memory _queryData
    ) external {
```

Tutorial: Liquidator



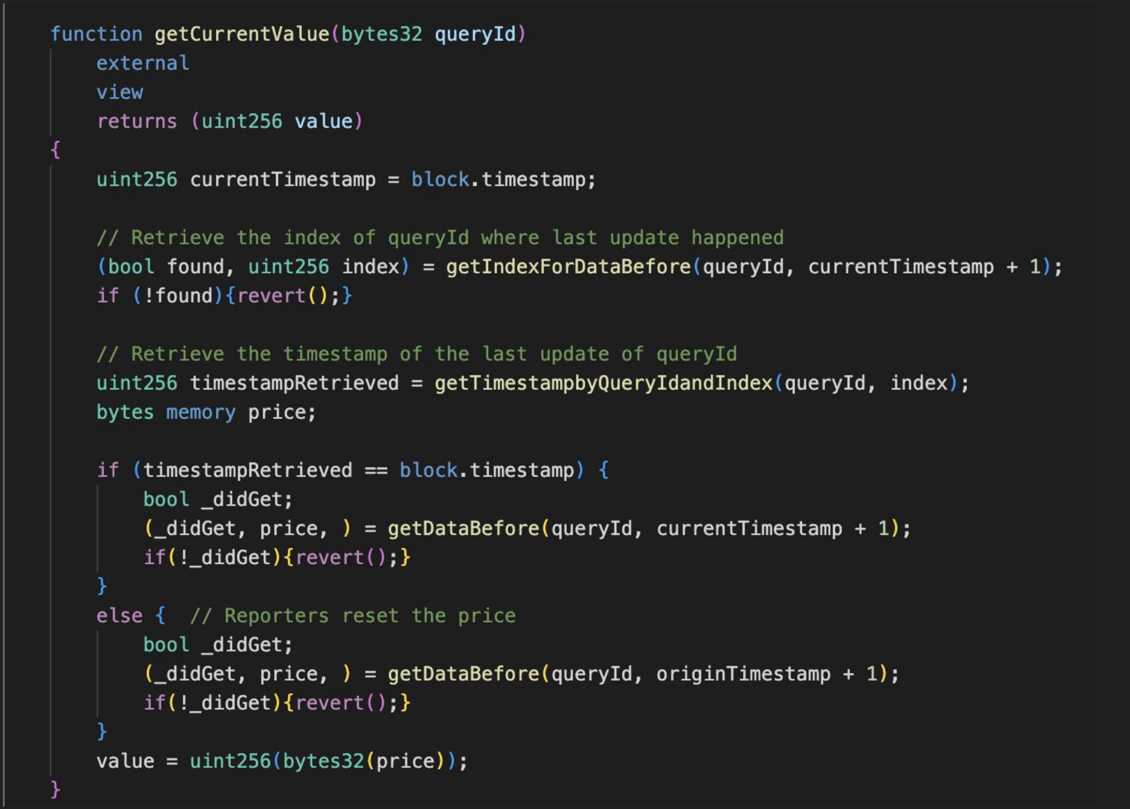
```
contract Liquidator{
    bytes32 queryId;
    address public player;
    address public tokenX;
    address public tokenUSD;
    ISimpleTellor public priceFeedOracle;

    constructor(address _player, address _tokenX, address _tokenUSD, address tellor){
        player = _player;
        tokenX = _tokenX;
        tokenUSD = _tokenUSD;
        queryId = keccak256(bytes("TokenX"));
        priceFeedOracle = ISimpleTellor(tellor);
    }

    function completed() external view returns (bool) {
        return IERC20(tokenUSD).balanceOf(player) == 10000;
    }

    function exchange(uint256 amountIn) external returns (bool success)
    {
        require(IERC20(tokenX).transferFrom(msg.sender, address(this), amountIn), 'transferFrom failed, is the bank approved?');
        uint256 _value = priceFeedOracle.getCurrentValue(queryId);
        require(_value != 0, "price of a token in USD cannot be 0");
        uint256 amountOut = amountIn * _value;
        require(IERC20(tokenUSD).transfer(msg.sender, amountOut), "transfer to reciepent failed");
        success = true;
        }
}
```

Set price feed oracle to SimpleTellor

Calls getCurrentValue of price feed oracle

Tutorial: Liquidator

SimpleTellor oracle



```
function getCurrentValue(bytes32 queryId)
    external
    view
    returns (uint256 value)
{
    uint256 currentTimestamp = block.timestamp;

    // Retrieve the index of queryId where last update happened
    (bool found, uint256 index) = getIndexForDataBefore(queryId, currentTimestamp + 1);
    if (!found){revert();}

    // Retrieve the timestamp of the last update of queryId
    uint256 timestampRetrieved = getTimestampbyQueryIdandIndex(queryId, index);
    bytes memory price;

    if (timestampRetrieved == block.timestamp) {
        bool _didGet;
        (_didGet, price, ) = getDataBefore(queryId, currentTimestamp + 1);
        if(!_didGet){revert();}
    }
    else {  // Reporters reset the price
        bool _didGet;
        (_didGet, price, ) = getDataBefore(queryId, originTimestamp + 1);
        if(!_didGet){revert();}
    }
    value = uint256(bytes32(price));
}
```

- Get the current price feed using queryId

Thank You