Workshop #3

Secure Development by OpenZeppelin

08/26 - 12PM PST / 7PM UTC

The Dangers of Price Oracles in Smart Contracts

Martin Abbatemarco

Security Researcher at OpenZeppelin

REGISTRATION REQUIRED LIMITED TO 50 ATENDEES

Series of sessions

Secure Development

The dangers of token integration  

Strategies for secure access controls  

The dangers of price oracles

Secure smart contract upgrades

The perils of low-level smart contract code

. . .

Problem(s) ?

(2 minutes)

```solidity
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

interface IOracle {
    function getPrice(IERC20 token) external returns (uint256);
}

contract Example is Ownable {

    IERC20 public immutable token;
    IOracle public oracle;
    mapping(address => uint256) public deposits;

    constructor (address tokenAddress, address oracleAddress) {
        token = IERC20(tokenAddress);
        oracle = IOracle(oracleAddress);
    }

    function setOracle(address oracleAddress) external onlyOwner {
        oracle = IOracle(oracleAddress);
    }

    /// @notice Allows taking out tokens by first depositing twice their value in ETH
    /// @param amount amount of tokens to be taken
    function borrow(uint256 amount) external payable {
        uint256 depositRequired = amount * oracle.getPrice(token) * 2;
        require(msg.value == depositRequired, "Bad");

        token.transfer(msg.sender, amount);

        deposits[msg.sender] += depositRequired;
    }

    // [...]
}
```

some initial triggers

```solidity
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

interface IOracle {
    function getPrice(IERC20 token) external returns (uint256);
}

contract Example is Ownable {

    IERC20 public immutable token;
    IOracle public oracle;
    mapping(address => uint256) public deposits;

    constructor (address tokenAddress, address oracleAddress) {
        token = IERC20(tokenAddress);
        oracle = IOracle(oracleAddress);
    }

    function setOracle(address oracleAddress) external onlyOwner {
        oracle = IOracle(oracleAddress);
    }

    /// @notice Allows taking out tokens by first depositing twice their value in ETH
    /// @param amount amount of tokens to be taken
    function borrow(uint256 amount) external payable {
        uint256 depositRequired = amount * oracle.getPrice(token) * 2;
        require(msg.value == depositRequired, "Bad");

        token.transfer(msg.sender, amount);

        deposits[msg.sender] += depositRequired;
    }

    // [...]
}
```

No events ?

Lack of docstrings ?

Unfriendly error message ?

What kind of tokens ?

Reentrancy ?

Who sets the oracle's address ?

What's the process to set it ? What's behind the onlyOwner ? Are those powers documented ?

What are the risks if ownership is compromised ?

and wait, about that oracle thing...

about that oracle thing...

```
uint256 depositRequired = amount * oracle.getPrice(token) * 2;
```

What if the price is zero ? What if the price is absurdly large ?

What's the price unit ? What if the price is inverted ?

How many decimals does the price have ?

How's the price calculated in the oracle ?

Can I actually read the price on-chain ? Any auth required beforehand ?

How's the price updated in the oracle ? How frequently ? Delays ?

What if the price of the oracle cannot be updated ?

Can the oracle's logic be upgraded ? Who ? How ? When ?

Are there privileged roles in the oracle ? What are their powers ?

Can it be manipulated ? Can it be shut down ?

If so, how ? who can do it ? when ?

Are there fallback oracles to use as safety nets ?

[...]

integrity

and

availability

```
oracle.getPrice(...)
```

want the perfect solution ?

there’s not

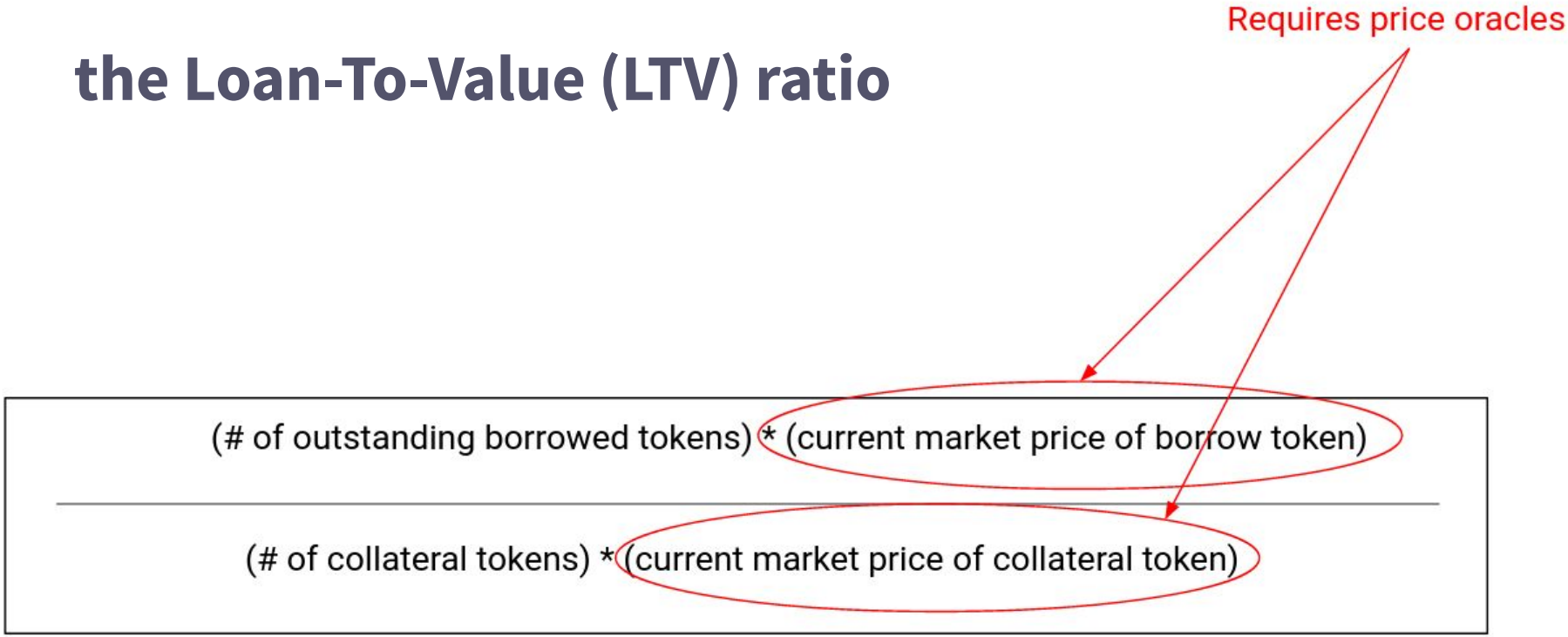
understanding risks

to make informed decisions

common use case of a price oracle

https://forum.openzeppelin.com/t/introduction-to-the-overcollateralized-loan-pattern-defi-primitive-and-its-security-considerations/2141

the Loan-To-Value (LTV) ratio



Slide shamelessly copied from Austin Williams' presentation

what if the oracle is manipulated ?

the compromised price oracle case

Buy 10 kitties

Give away kitties

Project selling kitties

Get price in ETH

Price oracle

the compromised price oracle case

Buy ALL kitties

Give away ALL kitties for free

Project selling kitties

Get price in ETH

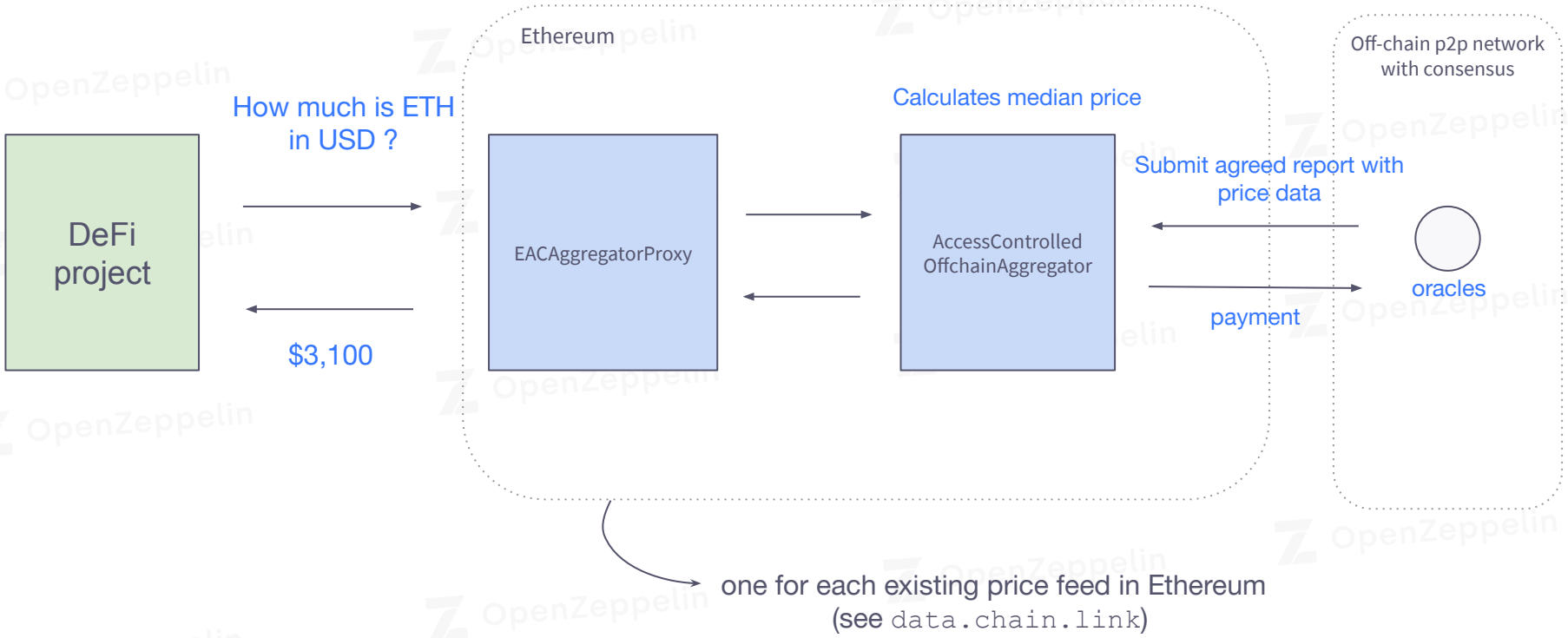
0 ETH

Compromised price oracle

the compromised price oracle case

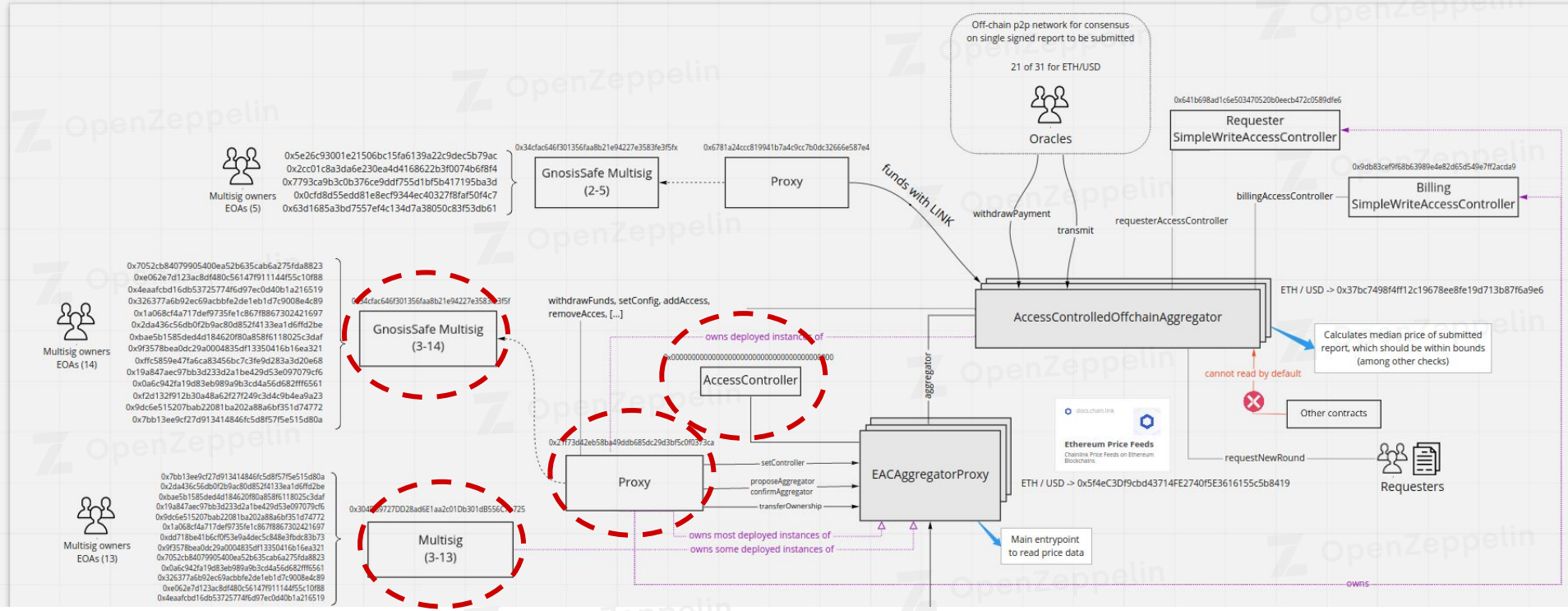
alternatives

ChainLink price feeds


DeFi project
How much is ETH in USD ?
$3,100
Ethereum
EACAggregatorProxy
Calculates median price
AccessControlled OffchainAggregator
Off-chain p2p network with consensus
Submit agreed report with price data
oracles
payment
one for each existing price feed in Ethereum
(see data.chain.link)

reality is a bit more complicated

(and this is still simplified)



what are their powers ?

reading a price

(according to the official docs)

```
AggregatorV3Interface(0x...).latestRoundData()

AggregatorV3Interface(0x...).getRoundData(uint80)
```

```
AggregatorV2V3Interface(0x...).latestAnswer()

AggregatorV2V3Interface(0x...).getAnswer(uint256)
```

reading a price

(according to the official docs)

```
AggregatorV3Interface(0x...).latestRoundData()

AggregatorV3Interface(0x...).getRoundData(uint80)
```

→ **Denial of Service ?**

```
AggregatorV2V3Interface(0x...).latestAnswer()

AggregatorV2V3Interface(0x...).getAnswer(uint256)
```

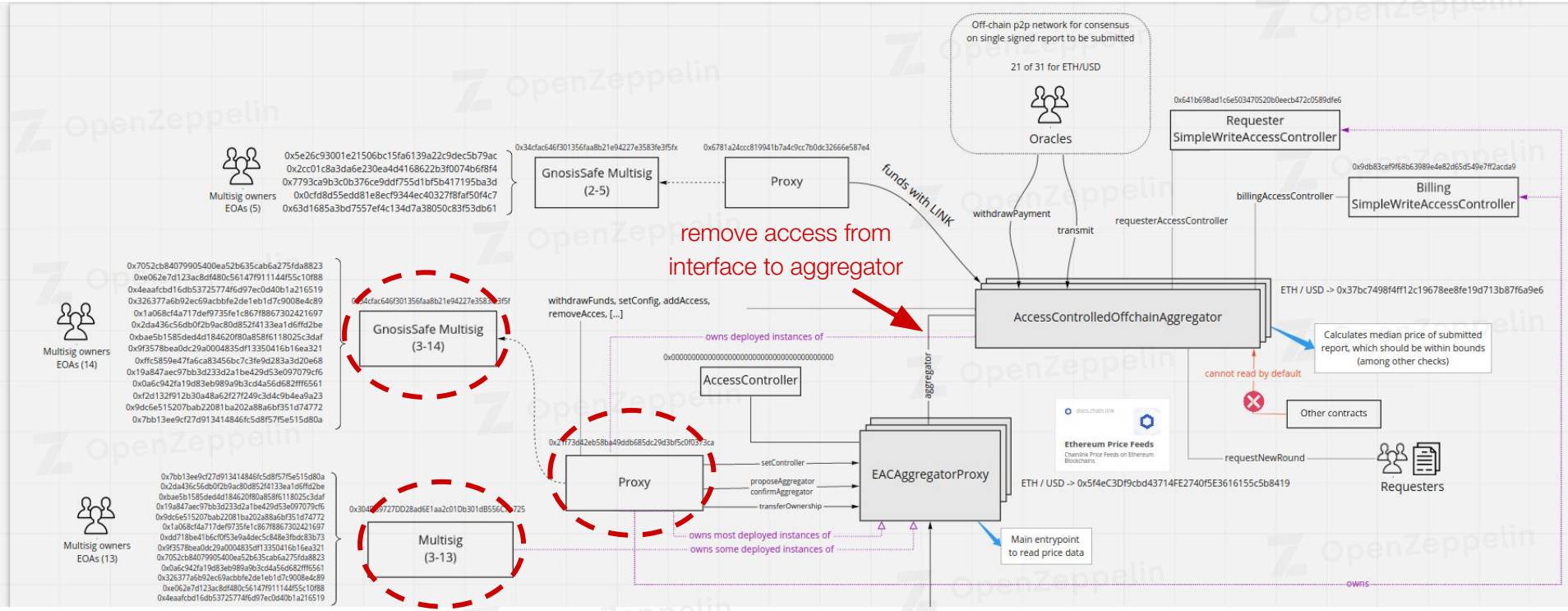
→ **Deprecated**

→ **Can return zero price**

→ **Denial of Service ?**

reality is a bit more complicated

(and this is still simplified)



What other powers can you find ?

if interface cannot access aggregator

```
AggregatorV3Interface(0x...).latestRoundData()
AggregatorV3Interface(0x...).getRoundData(uint80)
```


```
AggregatorV2V3Interface(0x...).latestAnswer()
AggregatorV2V3Interface(0x...).getAnswer(uint256)
```

querying a price, defensively

```
function getPrice(address priceFeedAddress) external view returns (int256) {
    try AggregatorV3Interface(priceFeedAddress).latestRoundData() returns (
        uint80,
        int256 price,
        uint256,
        uint256,
        uint80
    ) {
        return price;
    } catch Error(string memory) {
        // handle failure here:
        // revert, call propietary fallback oracle, fetch from another 3rd-party oracle, etc.
    }
}
```

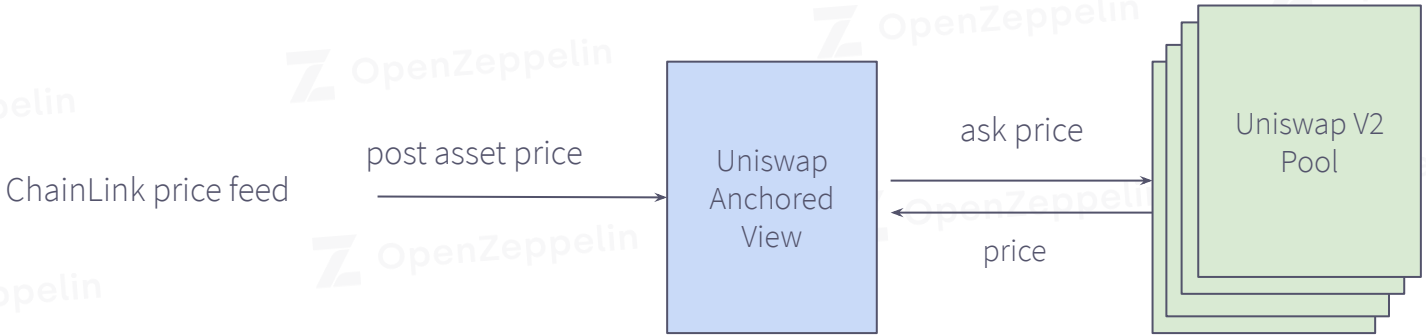
```
function getPrice(address priceFeedAddress) external view returns (int256) {
    try AggregatorV2V3Interface(priceFeedAddress).latestAnswer() returns (int256 price) {
        if(price > 0) {
            return price;
        } else {
            // `latestAnswer` is a deprecated method to read prices, yet still used in the wild.
            // It can return zero under certain circumstances, so integrations should handle this case.
            // Either with revert, call propietary fallback oracle, fetch from another 3rd-party oracle, etc.
        }
    } catch Error(string memory) {
        // handle failure here
        // revert, call propietary fallback oracle, fetch from another 3rd-party oracle, etc.
    }
}
```

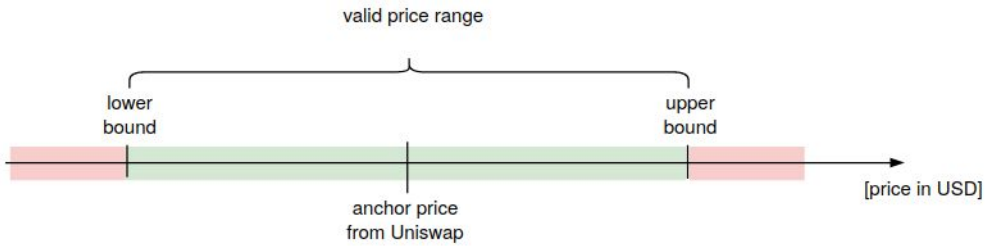
some remarks and recommendations

- Don’t assume decentralized and permissionless systems. Better verify roles and powers.
- You’re trusting off-chain operators and multisigs (listed at data.chain.link)
- Code defensively to mitigate potential threats.
- Don’t use deprecated interfaces. Understand subtleties of recommended ones.
- Depending on interfaces, check whether prices can be zero.
- Check units and decimals of each price feed.

The Open Price Feed

compound.finance/docs/prices


ChainLink price feed
post asset price
Uniswap Anchored View
ask price
price
Uniswap V2 Pool


valid price range
lower bound
upper bound
anchor price from Uniswap
[price in USD]

How does this happen ?

ChainLink price feed

post asset price

Are there privileged roles here ?

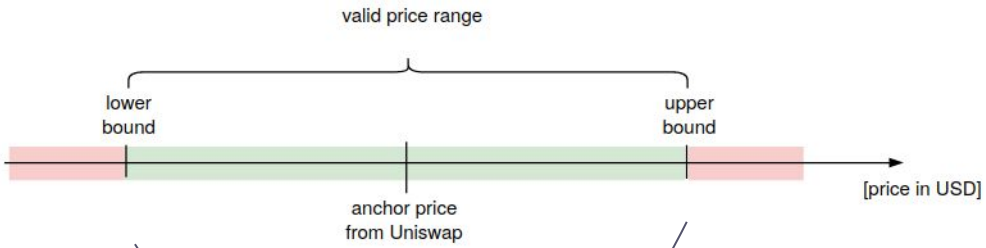
Uniswap Anchored View

ask price

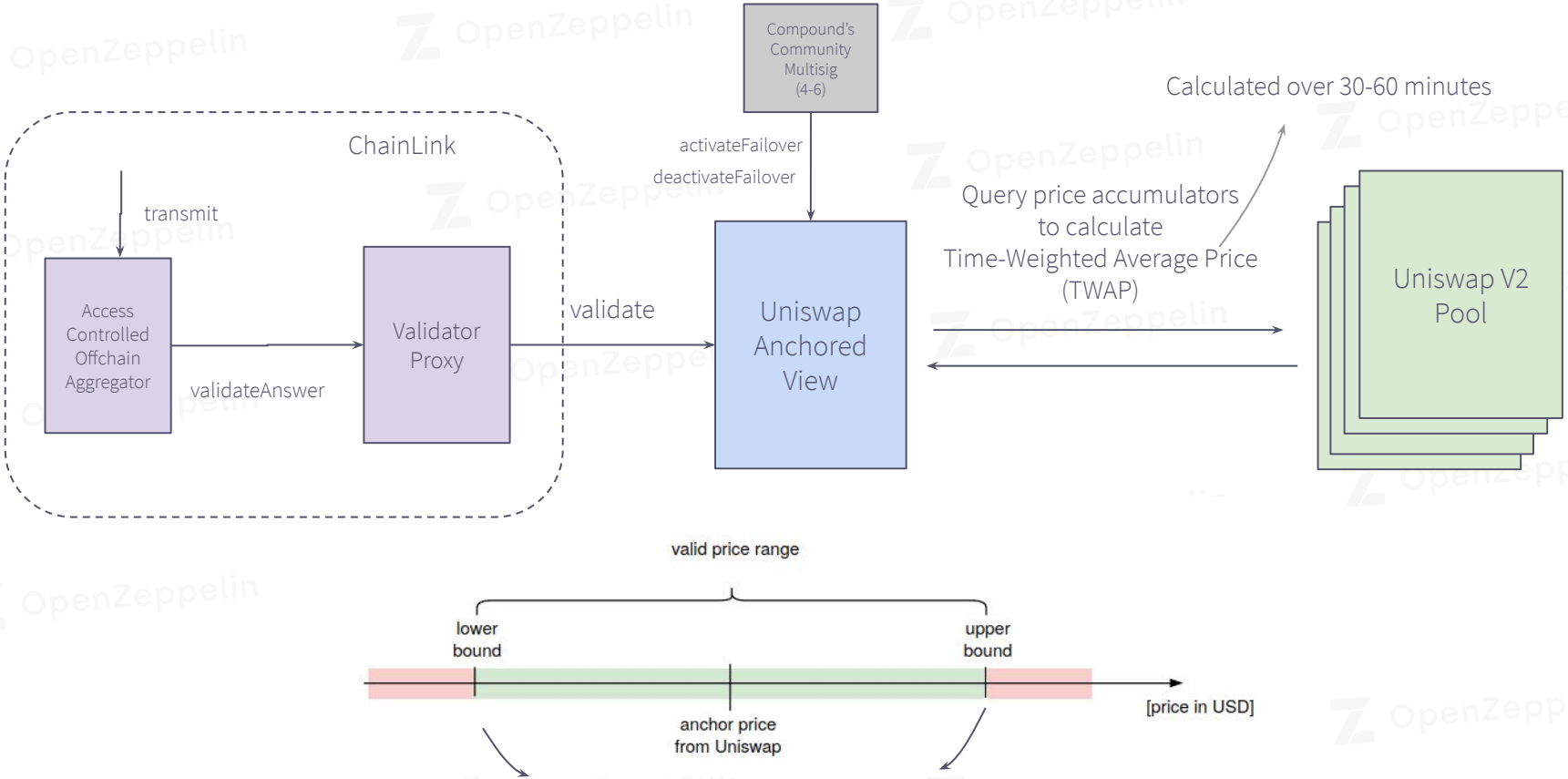
price

How's this price calculated ?

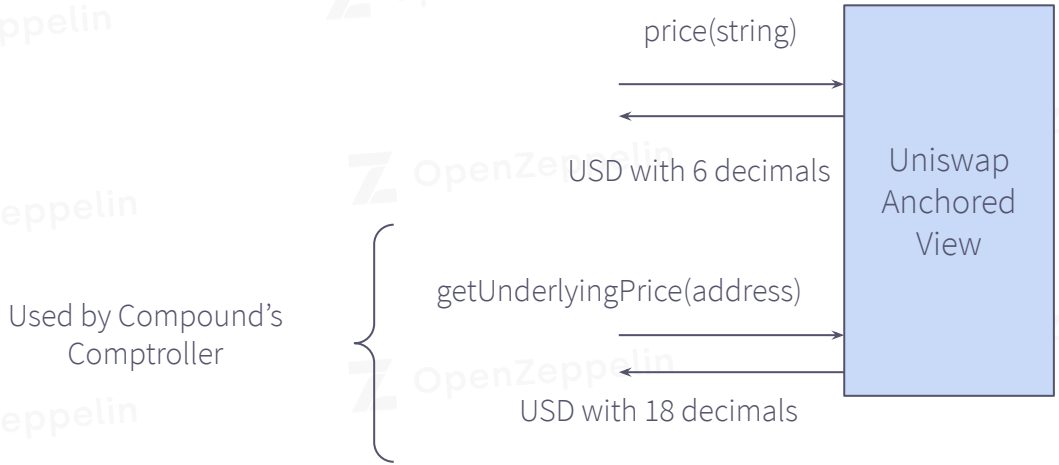
Uniswap V2 Pool



Who sets these bounds ?


Compound's Community Multisig (4-6)
activateFailover
deactivateFailover
Calculated over 30-60 minutes
ChainLink
transmit
Access Controlled Offchain Aggregator
validateAnswer
Validator Proxy
validate
Uniswap Anchored View
Query price accumulators to calculate Time-Weighted Average Price (TWAP)
Uniswap V2 Pool
valid price range
lower bound
upper bound
anchor price from Uniswap
[price in USD]
Set at deployment. Immutable % of the anchor price.

reading a price



some remarks and recommendations

- ChainLink powers are limited by TWAP anchor.
- Failover mode for problems in reporting. In failover, might need to update price first.
- Period and bounds are the same for all assets.
- Careful with the amount of decimals. Depends on function queried.
- Queries revert for unsupported symbol / cToken address.
- USDT, TUSD and USDC are not reported. Assumed pegged to USD.
- Stay up to date with changes. After upgrades (usually in Compound Protocol) views might become outdated and unreported.

Uniswap TWAPs

what you shouldn't do

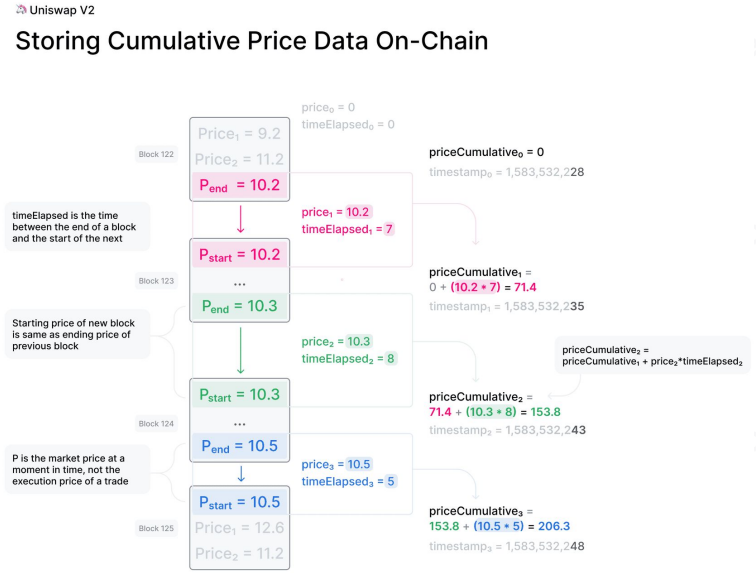
Pool of 2 assets

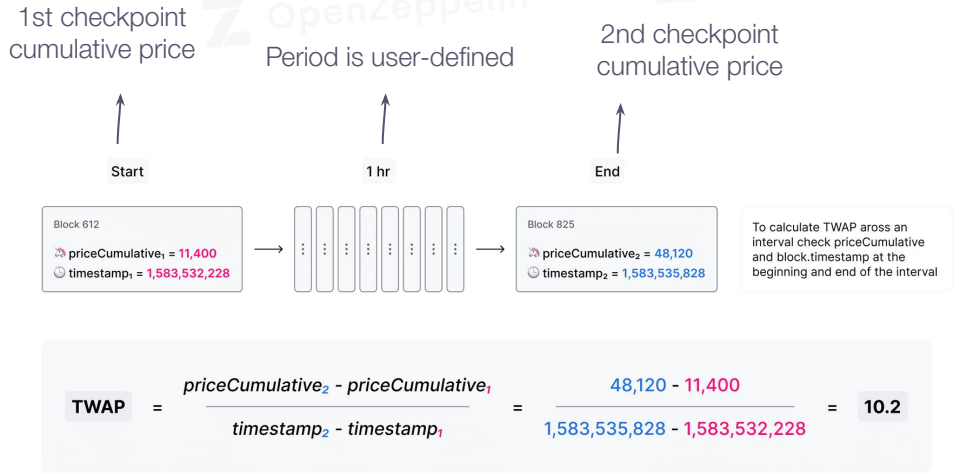
1 WETH is 3 DAI

$$\frac{\text{dai.balanceOf(pool)}}{\text{weth.balanceOf(pool)}} = 3$$

can be manipulated

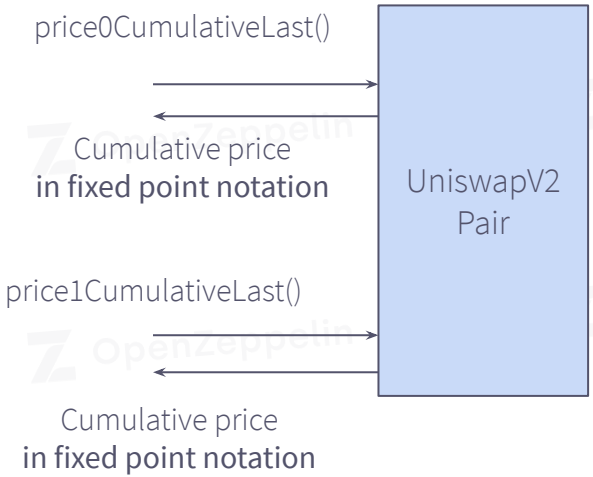
Uniswap V2 Time-Weighted Average Prices





uniswap.org/blog/uniswap-v2/#price-oracles

Uniswap V2 Time-Weighted Average Prices



Uniswap V2 Time-Weighted Average Prices

You can use the UniswapV2OracleLibrary

```
// library with helper methods for oracles that are concerned with computing average prices
library UniswapV2OracleLibrary {
    using FixedPoint for *;

    // helper function that returns the current block timestamp within the range of uint32, i.e. [0, 2**
    function currentBlockTimestamp() internal view returns (uint32) {
        return uint32(block.timestamp % 2 ** 32);
    }

    // produces the cumulative price using counterfactuals to save gas and avoid a call to sync.
    function currentCumulativePrices(
        address pair
    ) internal view returns (uint price0Cumulative, uint price1Cumulative, uint32 blockTimestamp) {
```

https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/libraries/UniswapV2OracleLibrary.sol

You can see an example of a 24-hour TWAP oracle

You can see it used as anchor in the Open Price feed with a rolling-window mechanism

Uniswap V2 Time-Weighted Average Prices

You can use the UniswapV2OracleLibrary

You can see an example of a 24-hour TWAP oracle

```
// fixed window oracle that recomputes the average price for the entire period once every period
// note that the price average is only guaranteed to be over at least 1 period, but may be over a longer period
contract ExampleOracleSimple {
    using FixedPoint for *;

    uint public constant PERIOD = 24 hours;

    IUniswapV2Pair immutable pair;
    address public immutable token0;
    address public immutable token1;

    uint    public price0CumulativeLast;
    uint    public price1CumulativeLast;
    uint32  public blockTimestampLast;
    FixedPoint.uq112x112 public price0Average;
    FixedPoint.uq112x112 public price1Average;
```

github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/examples/ExampleOracleSimple.sol

You can see it used as anchor in the Open Price feed with a rolling-window mechanism

Uniswap V2 Time-Weighted Average Prices

You can use the UniswapV2OracleLibrary

You can see an example of a 24-hour TWAP oracle

See `fetchAnchorPrice` and `pokeWindowValues` functions

```
/**
 * @dev Fetches the current token/usd price from uniswap, with 6 decimals of precision.
 * @param conversionFactor 1e18 if seeking the ETH price, and a 6 decimal ETH-USDC price in the case of other assets
 */
function fetchAnchorPrice(bytes32 symbolHash, TokenConfig memory config, uint conversionFactor) internal virtual returns (uint) {
```

You can see it used as anchor in the Open Price feed with a rolling-window mechanism

```
/**
 * @dev Get time-weighted average prices for a token at the current timestamp.
 *  Update new and old observations of lagging window if period elapsed.
 */
function pokeWindowValues(TokenConfig memory config) internal returns (uint, uint, uint) {
```

https://etherscan.io/address/0x6d2299c48a8dd07a872fdd0f8233924872ad1071#code

some remarks and considerations

- Tradeoffs in length of time period
  - Ease of manipulation vs. accuracy during high volatility
- In Uniswap v2, the TWAP of A in B is not the reciprocal of B in A
  - That’s why there are two accumulators. Query the one you need.
- Beware of units and decimals of returned prices
- Use available libraries and utilities in Uniswap’s repository
- More details at “Oracle Integrity” section in uniswap.org/audit.html

what about Uniswap V3 ?

in Uniswap V3

no more checkpoints of accumulators

geometric mean instead of arithmetic mean

single accumulator tracking $a_t = \sum_{i=1}^{t} \log_{1.0001}(P_i)$

For simple implementations

```
/// @title Oracle library
/// @notice Provides functions to integrate with V3 pool oracle
library OracleLibrary {
    /// @notice Fetches time-weighted average tick using Uniswap V3 oracle
    /// @param pool Address of Uniswap V3 pool that we want to observe
    /// @param period Number of seconds in the past to start calculating time-weighted average
    /// @return timeWeightedAverageTick The time-weighted average tick from (block.timestamp - period) to blo
    function consult(address pool, uint32 period) internal view returns (int24 timeWeightedAverageTick) {
```

github.com/Uniswap/uniswap-v3-periphery/blob/main/contracts/libraries/OracleLibrary.sol

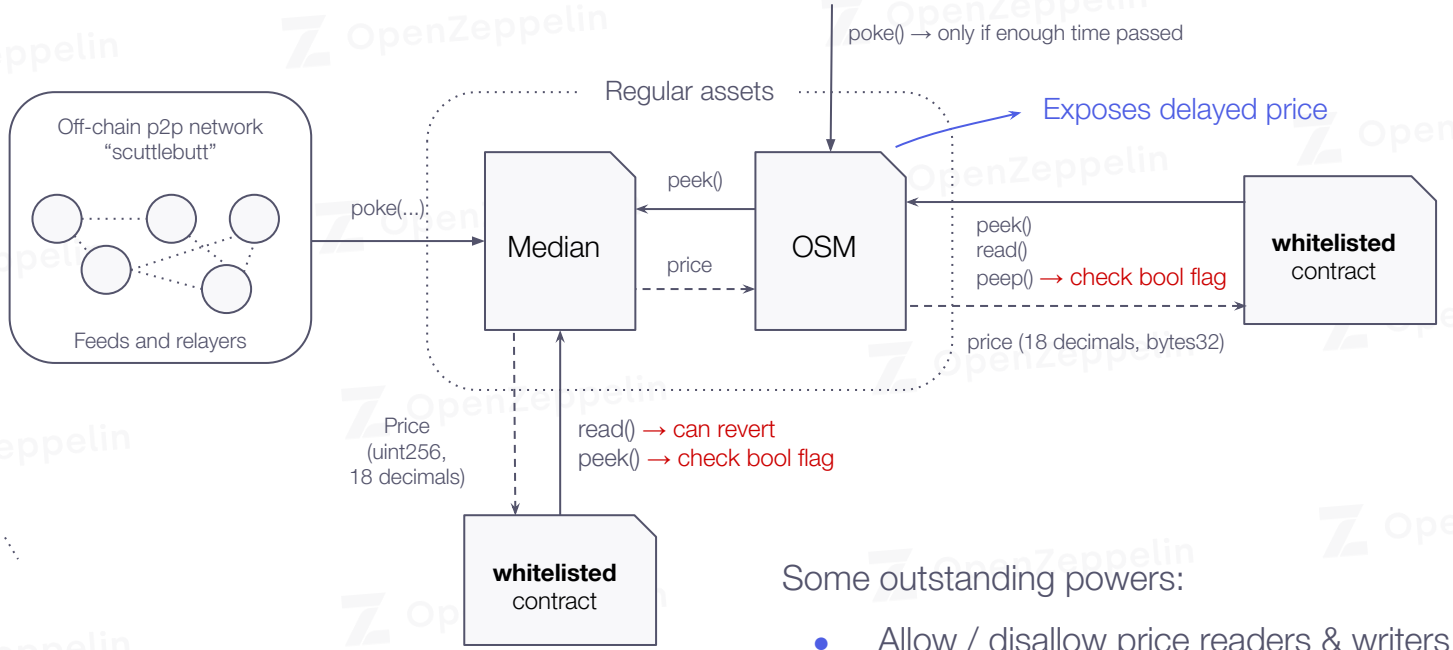
reverts if period's too big

can be negative
+
need to transform tick value to price!

docs.uniswap.org/protocol/concepts/V3-overview/oracle#deriving-price-from-a-tick

Maker Oracles

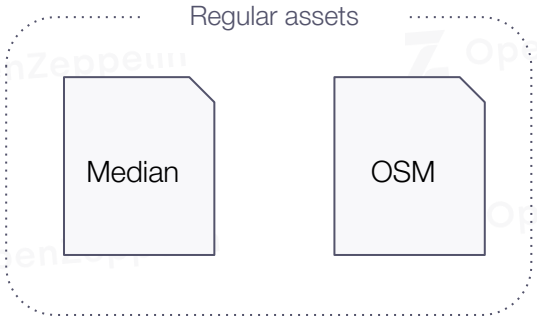
not everyone can access these on-chain



Some outstanding powers:

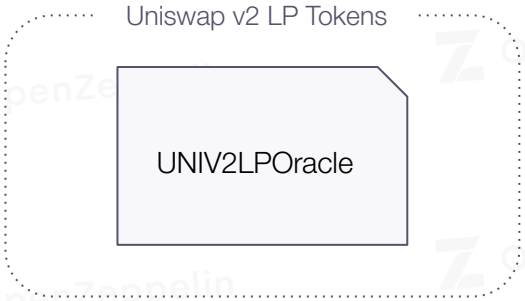
- Allow / disallow price readers & writers
- Median → Change threshold to update price
- OSM → stop, change delay, delete prices

https://docs.makerdao.com/smart-contract-modules/oracle-module

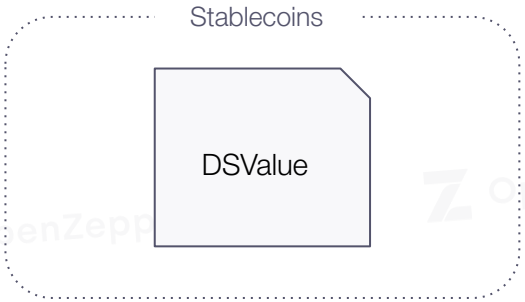


https://github.com/makerdao/median

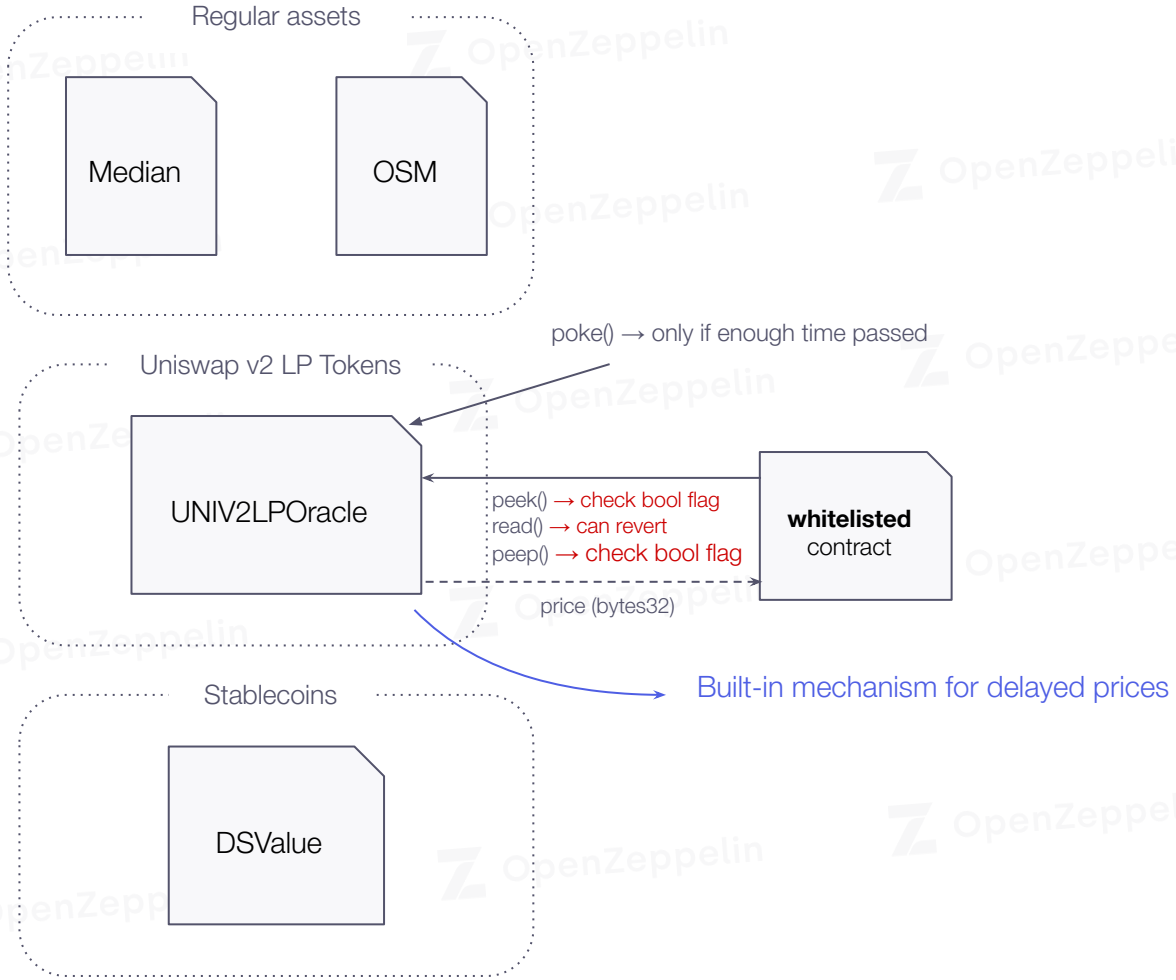
https://github.com/makerdao/osm

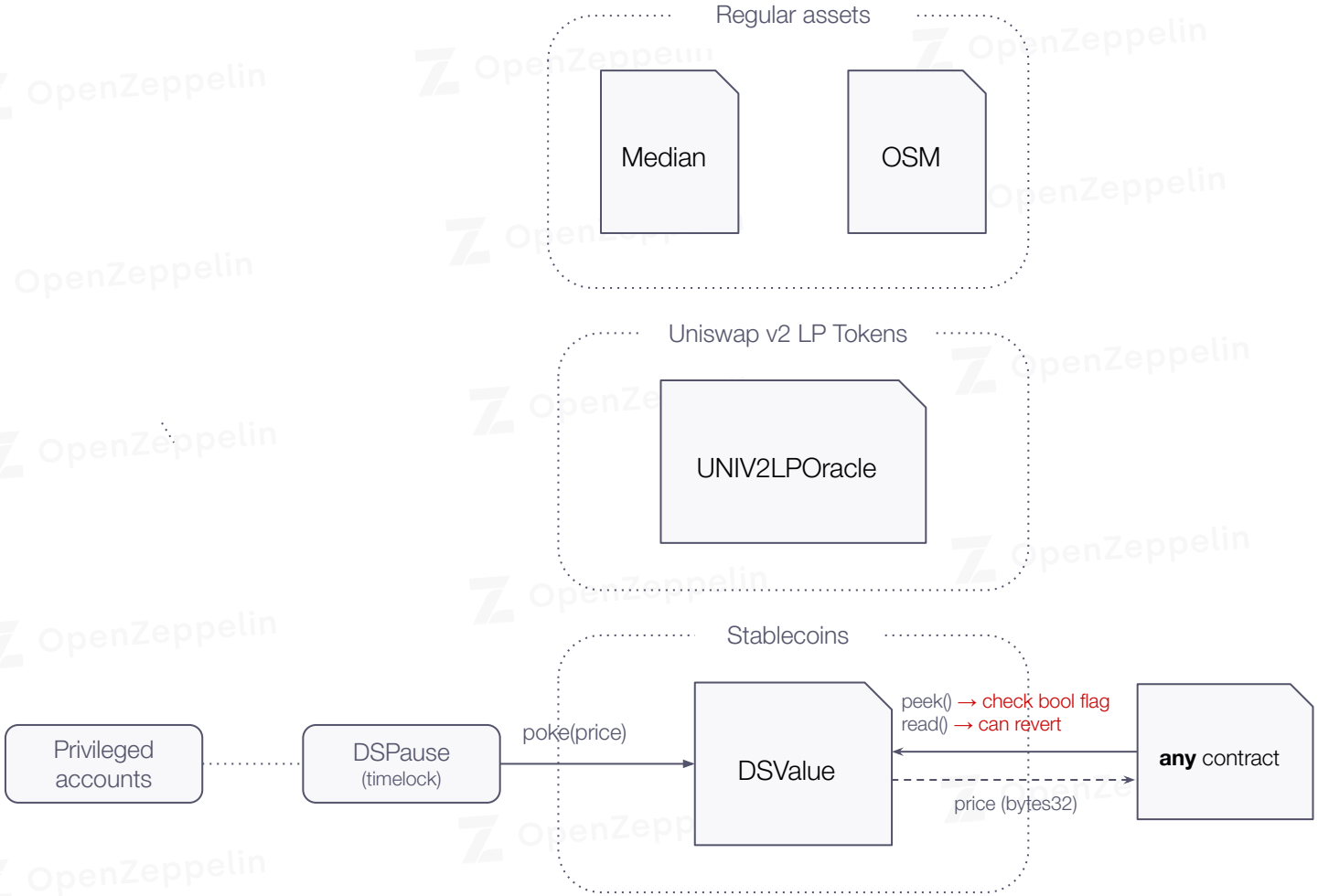


https://github.com/makerdao/univ2-lp-oracle



https://github.com/dapphub/ds-value


Regular assets
Median
OSM
poke() → only if enough time passed
Uniswap v2 LP Tokens
UNIV2LPOracle
peek() → check bool flag
read() → can revert
peep() → check bool flag
whitelisted
contract
price (bytes32)
Built-in mechanism for delayed prices
Stablecoins
DSValue


Regular assets
Median
OSM
Uniswap v2 LP Tokens
UNIV2LPOracle
Stablecoins
Privileged accounts
DSPause (timelock)
poke(price)
DSValue
peek() → check bool flag
read() → can revert
any contract
price (bytes32)

On using price oracles

Closing thoughts

1. No silver bullet. There are tradeoffs.
2. To different extents trusting owners, nodes, operators, arbitrageurs.
3. Defensive coding. Delays, fallback oracles, might also reduce risks.
4. Oracles are a living thing. Participate. Contribute. Stay up to date.
5. Read. Learn. Take better informed and conscious decisions.

On price oracles

Where do I learn more ?

- ethereum.org/en/developers/docs/oracles
- makerdao.world/en/faqs/oracles
- docs.uniswap.org/protocol/concepts/V3-overview/oracle
- docs.umaproject.org/oracle/econ-architecture
- shouldiusespotpriceasmyoracle.com
- samczsun.com/so-you-want-to-use-a-price-oracle

Series of sessions

Secure Development

The dangers of token integration 

Strategies for secure access controls 

The dangers of price oracles 

Secure smart contract upgrades

The perils of low-level smart contract code

. . .

In the meantime…

docs.openzeppelin.com/defender/advisor

We're hiring!

Open Roles

- Blockchain Security Engineer
- Full Stack Ethereum Developer
- Open Source Developer
- Site Reliability Engineer

Check out more

zpl.in/join

Thanks!

Learn more

openzeppelin.com
defender.openzeppelin.com
blog.openzeppelin.com
forum.openzeppelin.com

Contact

@tinchoabbate
tincho@openzeppelin.com