ABDK CONSULTING

SMART CONTRACT AUDIT

USM

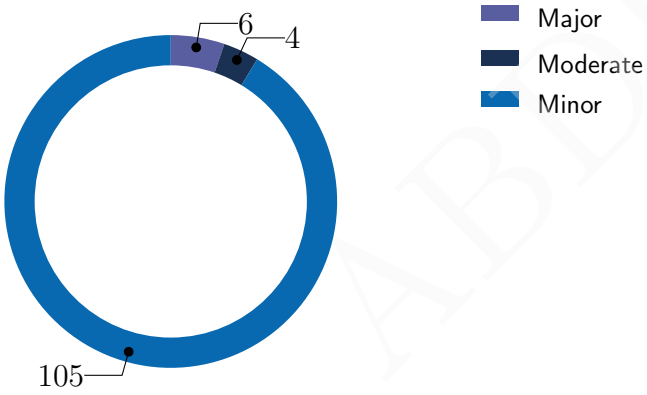
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
03/12/21

We've been asked to review the USM smart contracts that support the FUM protocol. We have identified a few issues of major and moderate severity, and a number of minor important ones.



Findings

ID	Severity	Subject	Status
CVF-1	Minor	Bad naming	Opened
CVF-2	Minor	Improper access specifiers	Opened
CVF-3	Minor	Bad naming	Opened
CVF-4	Minor	Two version of same structure	Opened
CVF-5	Minor	Redundant passing	Opened
CVF-6	Minor	Improper comment	Opened
CVF-7	Minor	Improper comment	Opened
CVF-8	Minor	Improper approach	Opened
CVF-9	Minor	Logging missing	Opened
CVF-10	Major	Returned value missing	Opened
CVF-11	Minor	Complicated code	Opened
CVF-12	Minor	Code duplication	Opened
CVF-13	Minor	Improper check placement	Opened
CVF-14	Minor	Code duplication	Opened
CVF-15	Minor	Complicated code	Opened
CVF-16	Minor	Redundant function	Opened
CVF-17	Minor	Redundant function	Opened
CVF-18	Minor	Redundant function	Opened
CVF-19	Minor	Single responsibility principle violation	Opened
CVF-20	Minor	Complicated code	Opened
CVF-21	Minor	Overflow	Opened
CVF-22	Minor	Redundant check	Opened
CVF-23	Minor	Improper comment	Opened
CVF-24	Minor	Improper approach	Opened
CVF-25	Minor	Improper comment	Opened
CVF-26	Minor	Inconsistent terminology	Opened
CVF-27	Moderate	Improper approach	Opened

ID	Severity	Subject	Status
CVF-28	Minor	Improper approach	Opened
CVF-29	Minor	Improper approach	Opened
CVF-30	Minor	Improper rounding	Opened
CVF-31	Minor	Precision missing	Opened
CVF-32	Minor	Precision missing	Opened
CVF-33	Minor	Precision missing	Opened
CVF-34	Minor	Precision missing	Opened
CVF-35	Minor	Improper placement of event definition	Opened
CVF-36	Minor	Similar functions	Opened
CVF-37	Minor	Redundant function	Opened
CVF-38	Minor	Unclear semantic	Opened
CVF-39	Minor	Improper implementation	Opened
CVF-40	Minor	Function missing	Opened
CVF-41	Minor	Bad naming	Opened
CVF-42	Minor	Improper access specifiers	Opened
CVF-43	Minor	Complicated code	Opened
CVF-44	Minor	Improper approach	Opened
CVF-45	Minor	Improper comment	Opened
CVF-46	Minor	Improper comment	Opened
CVF-47	Minor	Complicated code	Opened
CVF-48	Minor	Require missing	Opened
CVF-49	Minor	Complicated code	Opened
CVF-50	Minor	Redundant code	Opened
CVF-51	Minor	Bad naming	Opened
CVF-52	Minor	Complicated code	Opened
CVF-53	Minor	Bad naming	Opened
CVF-54	Minor	Improper access specifiers	Opened
CVF-55	Minor	Redundant check	Opened
CVF-56	Minor	Redundant code	Opened
CVF-57	Minor	Bad naming	Opened

ID	Severity	Subject	Status
CVF-58	Minor	Improper approach	Opened
CVF-59	Minor	Redundant check	Opened
CVF-60	Minor	Improper interface placement	Opened
CVF-61	Minor	Comment missing	Opened
CVF-62	Minor	Improper access specifiers	Opened
CVF-63	Minor	Bad naming	Opened
CVF-64	Minor	Comment missing	Opened
CVF-65	Minor	Constant missing	Opened
CVF-66	Minor	Improper approach	Opened
CVF-67	Minor	Improper approach	Opened
CVF-68	Minor	Improper approach	Opened
CVF-69	Minor	Check missing	Opened
CVF-70	Minor	Complicated code	Opened
CVF-71	Minor	Improper access specifier	Opened
CVF-72	Minor	Bad naming	Opened
CVF-73	Minor	Bad naming	Opened
CVF-74	Minor	Complicated code	Opened
CVF-75	Minor	Improper approach	Opened
CVF-76	Moderate	Underflow	Opened
CVF-77	Minor	File imported twice	Opened
CVF-78	Minor	Complicated code	Opened
CVF-79	Moderate	Improper approach	Opened
CVF-80	Minor	Improper comment	Opened
CVF-81	Minor	Complicated code	Opened
CVF-82	Major	Ignored returned value	Opened
CVF-83	Minor	Bad naming	Opened
CVF-84	Minor	Improper approach	Opened
CVF-85	Minor	Improper approach	Opened
CVF-86	Minor	Not logged events	Opened
CVF-87	Minor	Complicated code	Opened

ID	Severity	Subject	Status
CVF-88	Minor	Bad naming	Opened
CVF-89	Minor	Bad naming	Opened
CVF-90	Minor	Improper access specifier	Opened
CVF-91	Minor	Improper access specifier	Opened
CVF-92	Minor	ERC20 inconsistency	Opened
CVF-93	Minor	Bad naming	Opened
CVF-94	Minor	Complicated code	Opened
CVF-95	Minor	Improper approach	Opened
CVF-96	Minor	Improper approach	Opened
CVF-97	Minor	Complicated code	Opened
CVF-98	Minor	Improper approach	Opened
CVF-99	Minor	Bad naming	Opened
CVF-100	Minor	Two versions of smart contract	Opened
CVF-101	Minor	Binary shift instead of division	Opened
CVF-102	Moderate	Improper approach	Opened
CVF-103	Minor	Documentation comment missing	Opened
CVF-104	Minor	Typo	Opened
CVF-105	Minor	Improper approach	Opened
CVF-106	Minor	Complicated code	Opened
CVF-107	Minor	Duplicated code	Opened
CVF-108	Minor	Improper approach	Opened
CVF-109	Minor	Complicated code	Opened
CVF-110	Minor	Improper access specifiers	Opened
CVF-111	Major	Improper approach	Opened
CVF-112	Minor	Magic number	Opened
CVF-113	Major	Improper approach	Opened
CVF-114	Major	Improper approach	Opened
CVF-115	Major	Improper approach	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	Mar. 10, 2021	D. Khovratovich	Initial Draft
0.2	Mar. 11, 2021	D. Khovratovich	Minor Revision
1.0	Mar. 12, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

We audited the code at the release 0.3.0, concretely the following files:

- oracles/ChainlinkOracle.sol;
- oracles/CompoundOpenOracle.sol;
- oracles/MedianOracle.sol;
- oracles/Oracle.sol;
- oracles/OurUniswapV2TWAPOracle.sol;
- Address.sol;
- Delegable.sol;
- FUM.sol;
- IUSM.sol;
- MinOut.sol;
- Ownable.sol;
- Proxy.sol;
- USM.sol;
- USMView.sol;
- WadMath.sol;
- WithOptOut.sol.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 About Customer

USM is a decentralized stablecoin project.ye

2.3 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.4 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** USM.sol

Recommendation 1e18 would be more readable.

Listing 1: Bad naming

```
36 uint public constant WAD = 10 ** 18;
```

3.2 CVF-2 Improper access specifiers

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** USM.sol

Recommendation These constants doesn't need to be public.

Listing 2: Improper access specifiers

```
36 uint public constant WAD = 10 ** 18;
   uint public constant HALF_WAD = WAD / 2;
   uint public constant BILLION = 10 ** 9;
   uint public constant HALF_BILLION = BILLION / 2;
```

3.3 CVF-3 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** USM.sol

Recommendation 1e9 would be more readable.

Listing 3: Bad naming

```
38 uint public constant BILLION = 10 ** 9;
```

3.4 CVF-4 Two version of same structure

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description It is confusing to have two version of the same structure. Is saved gas really worth it? Anyway, as the goal is to fit the whole structure into a single storage slot, it would be enough to define only LoadedState structure, and two functions to pack/unpack LoadedState to/from a bytes32 value.

Listing 4: Two version of same structure

```
47 struct StoredState {

55 struct LoadedState {
```

3.5 CVF-5 Redundant passing

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description Passing "this" seems redundant, as FUM could just look at msg.sender in constructor.

Listing 5: Redundant passing

```
75 fum = new FUM(this, optedOut_);
```

3.6 CVF-6 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description Ether price specified in terms of tokens (i.e. in terms of USD), actually has 5 digits before the dot, and 2 digits after the dot, i.e. 12345.67.

Listing 6: Improper comment

```
146 * be parsed as the minimum Ether price accepted, with 2 digits
    ↪ before and 5 digits after the comma.
```

3.7 CVF-7 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** USM.sol

Description Actually, ETH price in USM terms is passed, not USM price.

Listing 7: Improper comment

```
169 * are '0000' then the next 7 will be parsed as the maximum USM
    ↪ price accepted, with 5 digits before and 2 digits after
    ↪ the comma.
```

3.8 CVF-8 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description Such may ways to burn makes the code overcomplicated and confuse users, while users still have many ways to loose tokens. **Recommendation** Consider providing a single "right" way to burn.

Listing 8: Improper approach

```
172 if (recipient == address(this) || recipient == address(fum) ||
    ↪ recipient == address(0)) {
```

3.9 CVF-9 Logging missing

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Description This branch does not log the Transfer event which may violate the ERC20 standard if this function is called within 'transfer'.

Listing 9: Logging missing

```
173 _burnUsm(sender, payable(sender), amount, MinOut.parseMinEthOut(
    ↪ amount));
```

3.10 CVF-10 Returned value missing

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Description The returned value is ignored here.

Listing 10: Returned value missing

```
175 super._transfer(sender, recipient, amount);
```

3.11 CVF-11 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** USM.sol

Description For TWAP oracle, update time could be the same, but the new price could be different and more relevant to the moment. Why not to use it in such case?

Listing 11: Complicated code

```
310 if (!priceChanged) {                    // If the price isn't
    ↪ fresher than our old one, scrap it and stick to the old
    ↪ one
```

3.12 CVF-12 Code duplication

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description The expression "ls.ethUsdPrice * adjustment / price" appears in the code two times.
Recommendation Consider calculating before the conditional statement, but only when adjustment != WAD.

Listing 12: Code dublictation

```
338 adjustment = WAD.wadMax(ls.ethUsdPrice * adjustment / price);

341 adjustment = WAD.wadMin(ls.ethUsdPrice * adjustment / price);
```

3.13 CVF-13 Improper check placement

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Recommendation These checks should be made in the very beginning of the function.

Listing 13: Check placement

```
362 require(ls.ethUsdPriceTimestamp <= type(uint32).max, "
    ↪ ethUsdPriceTimestamp overflow");

371 require(ls.buySellAdjustmentTimestamp <= type(uint32).max, "
    ↪ buySellAdjustmentTimestamp overflow");
```

3.14 CVF-14 Code duplication

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description This code block is very similar to the block that deals with priceToStore.
Recommendation Consider reducing code duplication by moving common code into a function.

Listing 14: Code dublictation

```
373 uint adjustmentToStore = ls.buySellAdjustment + HALF_BILLION;
unchecked { adjustmentToStore /= BILLION; }
if (adjustmentToStore != storedState.buySellAdjustment) {
    require(adjustmentToStore <= type(uint80).max, "
      ↪ buySellAdjustment overflow");
    unchecked { emit BuySellAdjustmentChanged(adjustmentToStore
      ↪ * BILLION); }
}
```

3.15 CVF-15 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Recommendation This could be written as: storedPrice = StoredPrice ({...});

Listing 15: Complicated code

```
380 (storedState.timeSystemWentUnderwater,
     storedState.ethUsdPriceTimestamp, storedState.ethUsdPrice,
     storedState.buySellAdjustmentTimestamp, storedState.
        ↪ buySellAdjustment) =
      (uint32(ls.timeSystemWentUnderwater),
       uint32(ls.ethUsdPriceTimestamp), uint80(priceToStore),
       uint32(ls.buySellAdjustmentTimestamp), uint80(
          ↪ adjustmentToStore));
```

3.16 CVF-16 Redundant function

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description This function looks redundant, as "oracle" variable is public.

Listing 16: Redundant function

```
396 function latestOraclePrice() public virtual override view
       ↪ returns (uint price, uint updateTime) {
```

3.17 CVF-17 Redundant function

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description This function looks redundant, as "fum" variable is public.

Listing 17: Redundant function

```
408 function fumTotalSupply() public override view returns (uint
       ↪ supply) {
```

3.18 CVF-18 Redundant function

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description This function seems redundant, as "storedState" variable is already public.

Listing 18: Redundant function

```
423 function timeSystemWentUnderwater() public override view returns
    ↪ (uint timestamp) {
```

3.19 CVF-19 Single responsibility principle violation

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description This function violates the single responsibility principle, as it does several things: loads state, calculates buy/sell adjustment, and queries current balance.
Recommendation Consider decomposing this functions.

Listing 19: Single responsibility principle violation

```
429 function loadState() public view returns (LoadedState memory ls)
    ↪ {
```

3.20 CVF-20 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Recommendation This logic would be much simpler in case bool would be used instead of WadMath.Round enum.

Listing 20: Complicated code

```
454 WadMath.Round downOrUp = (upOrDown == WadMath.Round.Down ?
    ↪ WadMath.Round.Up : WadMath.Round.Down);
```

3.21 CVF-21 Overflow

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** USM.sol

Description Conversion to int may overflow.

Listing 21: Overflow

```
455 buffer = int(ethInPool) - int(usmToEth(ethUsdPrice, usmSupply,
    ↪ downOrUp));
```

3.22 CVF-22 Redundant check

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description This check looks very odd taking into account that Solidity 0.8.x automatically prevents underflows. Does it refer to the possible conversion overflow? If so, it would be better to use safe case library.

Listing 22: Redundant check

```
456 require(buffer <= int(ethInPool), "Underflow error");
```

3.23 CVF-23 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** USM.sol

Recommendation Probably, this function should return zero only when both, USM supply and ETH in the pool are zeros. Non-zero USM supply together with zero ETH is actually infinite debt ratio.

Listing 23: Improper comment

```
462 * @return ratio Debt ratio (or 0 if there's currently 0 ETH in
    ↪ the pool/price = 0: these should never happen after launch
    ↪ )
```

3.24 CVF-24 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description A one-time seed funding logic is spread through the code.
Recommendation Consider doing initial funding in the constructor, to keep the rest of the code clear.

Listing 24: Improper approach

```
466 ratio = (ethPoolValueInUsd == 0 ? 0 : usmSupply.wadDivUp(
      ↪ ethPoolValueInUsd));

514 if (fumSupply == 0) {

644 if (ls.ethPool == 0) {
```

3.25 CVF-25 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** USM.sol

Description Here price is the USM price in ETH terms, while in other parts of the code price is ETH price in USM terms.
Recommendation Consider using "price" term consistently. If both meanings are needed, consider using different terms, such as "price" and "reversePrice".

Listing 25: Improper comment

```
488 * @notice Calculate the *marginal* price of USM (in ETH terms):
      ↪ that is, of the next unit, before the price start sliding.
```

3.26 CVF-26 Inconsistent terminology

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** USM.sol

Description This is very confusing.
Recommendation Consider using consistent terminology, i.e. always use "buy"/"sell" for the same sides.

Listing 26: Inconsistent terminology

```
495 // Apply the adjustment if (side == Buy and adj < 1), or (side
    ↪ == Sell and adj > 1). You may be thinking "Wait! I
// thought the way the adjustment worked was, an adj > 1 was
    ↪ applied when we're *buying,* not selling." And your
// understanding was correct: the catch is that here we're "
    ↪ buying" USM, which is economically like *selling* ETH.
```

3.27 CVF-27 Improper approach

- **Severity** Moderate
- **Status** Opened
- **Category** Unclear behavior
- **Source** USM.sol

Description The adjustment is applied to FUM price, while it supposed to be applied to ethUsdPrice. Applying to FUM price (that could be zero) doesn't make much sense.

Listing 27: Improper approach

```
524 price = price.wadMul(adjustment, upOrDown);
```

3.28 CVF-28 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Recommendation It would be more efficient, more precise, and more clear to just store a 1-second decay factor as a WAD number, and use exponentiation by squaring to raise this 1-second decay factor into the power of the number of seconds passed since the system goes underwater. For the half-life period of 1 day, the 1 second decay factor would be 0.999991977495368426. For the half-life period of 1 minute, the 1 second decay factor would be 0.988514020352896135.

Listing 28: Improper approach

```
562     uint numHalvings = (currentTime - timeSystemWentUnderwater_)
            ↪ .wadDivDown(MIN_FUM_BUY_PRICE_HALF_LIFE);
        uint decayFactor = numHalvings.wadHalfExp();

574 uint numHalvings = (currentTime - storedTime).wadDivDown(
        ↪ BUY_SELL_ADJUSTMENT_HALF_LIFE);
    uint decayFactor = numHalvings.wadHalfExp(10);
```

3.29 CVF-29 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USM.sol

Description There are much more efficient methods to calculate square root, than via logarithm and exponent.

Listing 29: Improper approach

```
602 adjShrinkFactor = ls.ethPool.wadDivDown(ethPool1).wadExp(
        ↪ HALF_WAD);
```

3.30 CVF-30 Improper rounding

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Recommendation The exponent should be rounded down, as is used in the divisor of a fraction rounded up.

Listing 30: Improper rounding

```
624 uint exponent = usmIn.wadMulUp(usmSellPrice0).wadDivUp(ls.
        ↪ ethPool);
```

3.31 CVF-31 Precision missing

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Recommendation Exponentiation here should round up.

Listing 31: Missing precicion

```
629 adjGrowthFactor = ls.ethPool.wadDivUp(ethPool1).wadExp(HALF_WAD)
    ↪ ;
```

3.32 CVF-32 Precision missing

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Recommendation The exponentiation and the division by 2 here should round up.

Listing 32: Missing precicion

```
672 adjGrowthFactor = ethPool1.wadDivUp(ls.ethPool).wadExp(
    ↪ effectiveFumDelta / 2);
```

3.33 CVF-33 Precision missing

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Recommendation The value of fumDelta should be rounded down. This applies to both, the division and the debt ratio calculation.

Listing 33: Missing precicion

```
706 uint fumDelta = WAD.wadDivUp(WAD - debtRatio(ls.ethUsdPrice, ls.
    ↪ ethPool, ls.usmTotalSupply));
```

3.34 CVF-34 Precision missing

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** USM.sol

Recommendation Exponentiation here should round down.

Listing 34: Missing precicion

```
733 adjShrinkFactor = ethPool1.wadDivUp(ls.ethPool).wadExp(fumDelta
    ↪ / 2);
```

3.35 CVF-35 Improper placement of event definition

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IUSM.sol

Description This abstract contract defines types and functions of the USM API, but doesn't define events.
Recommendation Consider moving event definitions from USM to IUSM.

Listing 35: Improper placement of event definition

```
7 contract IUSM is IERC20 {
```

3.36 CVF-36 Similar functions

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IUSM.sol

Description These two functions are very similar, which is confusing.
Recommendation Consider adding documentation comments regarding the difference between them.

Listing 36: Similar functions

```
17 function latestPrice() public virtual view returns (uint price,
     ↪ uint updateTime);
function latestOraclePrice() public virtual view returns (uint
     ↪ price, uint updateTime);
```

3.37 CVF-37 Redundant function

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** IUSM.sol

Description Why this function is here? There is "totalSupply" function in the "FUM" smart contract.

Listing 37: Redundant function

```
20 function fumTotalSupply() public virtual view returns (uint
     ↪ supply);
```

3.38 CVF-38 Unclear semantic

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** IUSM.sol

Description The semantics of this function is unclear from its name and signature.
Recommendation Consider adding documentation comment.

Listing 38: Unclear semantic

```
21 function buySellAdjustment() public virtual view returns (uint
   ↪ adjustment);
```

3.39 CVF-39 Improper implementation

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IUSM.sol

Description These pure functions looks more like utilities, rather than like parts of USM API.
Recommendation Consider either making them "view" to allow then depend on USM runtime configuration, or implementing them here, or just removing them from IUSM and leaving only in USM.

Listing 39: Improper implementation

```
24 function ethBuffer(uint ethUsdPrice, uint ethInPool, uint
   ↪ usmSupply, WadMath.Round upOrDown) public virtual pure
   ↪ returns (int buffer);
function debtRatio(uint ethUsdPrice, uint ethInPool, uint
   ↪ usmSupply) public virtual pure returns (uint ratio);
function ethToUsm(uint ethUsdPrice, uint ethAmount, WadMath.
   ↪ Round upOrDown) public virtual pure returns (uint usmOut);
function usmToEth(uint ethUsdPrice, uint usmAmount, WadMath.
   ↪ Round upOrDown) public virtual pure returns (uint ethOut);
function usmPrice(Side side, uint ethUsdPrice, uint debtRatio_)
   ↪ public virtual pure returns (uint price);
function fumPrice(Side side, uint ethUsdPrice, uint ethInPool,
   ↪ uint usmEffectiveSupply, uint fumSupply, uint adjustment)
   ↪ public virtual pure returns (uint price);
30 function checkIfUnderwater(uint usmActualSupply, uint ethPool_,
   ↪ uint ethUsdPrice, uint oldTimeUnderwater, uint currentTime
   ↪ ) public virtual pure returns (uint
   ↪ timeSystemWentUnderwater_, uint usmSupplyForFumBuys);
```

3.40 CVF-40 Function missing

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WadMath.sol

Recommendation It would be helpful to have a mulDiv function that calculates $x \cdot y/z$ in one go. In case all three values are WADs, this function would be more efficient than wadMul+wadDiv. It also may allow product to exceed 256 bits as described here

Listing 40: Function missing

```
9 WadMath {
```

3.41 CVF-41 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** WadMath.sol

Recommendation 1e18 would be more readable.

Listing 41: Bad naming

```
12 uint public constant WAD = 10 ** 18;
```

3.42 CVF-42 Improper access specifiers

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** WadMath.sol

Description These constants don't need to be public.

Listing 42: Improper access specifiers

```
12 uint public constant WAD = 10 ** 18;
   uint public constant WAD_MINUS_1 = WAD - 1;
   uint public constant WAD_OVER_10 = WAD / 10;
   uint public constant WAD_OVER_20 = WAD / 20;
   uint public constant HALF_TO_THE_ONE_TENTH = 933032991536807416;
   uint public constant LOG_2_WAD_SCALED =
      ↪ 1589615936535143698135326734483216740 75;    // log_2
      ↪ (10**18) * 2**121
   uint public constant LOG_2_E_SCALED_OVER_WAD =
      ↪ 3835341275459348170;                  // log_2(e) * 2**121 /
      ↪  10**18
```

3.43 CVF-43 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WadMath.sol

Recommendation You don't need two versions of mul and div for rounding up and down. As rounding happens on the very last division, you just need plain division with adjustable rounding mode like this: function divWithRounding (uint x, uint y, Round rounding) internal pure returns (uint) { return (rounding == Round.Up ? x + y - 1 : x) / y; } and then just use this function inside mul and div for the final division.

Listing 43: Complicated code

```
24 function wadMulDown(uint x, uint y) internal pure returns (uint
   ↪ z) {

29 function wadMulUp(uint x, uint y) internal pure returns (uint z)
   ↪  {

38 function wadDivDown(uint x, uint y) internal pure returns (uint
   ↪ z) {

42 function wadDivUp(uint x, uint y) internal pure returns (uint z)
   ↪  {
```

3.44 CVF-44 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WadMath.sol

Recommendation This could be calculated as: exp = wadPow(HALF_TO_THE_ONE_TENTH, powerInTenthsUnscaled % 10) » (powerInTenthsUnscaled / 10);

Listing 44: Improper approach

```
74 exp = wadPow(HALF_TO_THE_ONE_TENTH, powerInTenthsUnscaled);
```

3.45 CVF-45 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** WadMath.sol

Recommendation Should be "for odd n" rather then "for even x".

Listing 45: Improper comment

```
90 *   and applying the equation for even x gives
   *   x^n = x * (x^2)^((n-1) / 2).
```

3.46 CVF-46 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** WadMath.sol

Description This is true only for odd n.

Listing 46: Improper comment

```
93 * Also, EVM division is flooring and floor[(n-1) / 2] = floor[n
   ↪ / 2].
```

3.47 CVF-47 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WadMath.sol

Recommendation This function could be optimized by unrolling the loop. Also, for those cases when x is known at compile time, such as when x is HALF_TO_THE_ONE_TENTH, the values $x\hat{2}$, $x\hat{4}$, etc could be pre-calculated.

Listing 47: Complicated code

```
95 function wadPow(uint x, uint n) internal pure returns (uint z) {
```

3.48 CVF-48 Require missing

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** WadMath.sol

Recommendation For x=0 logarithm is undefined, so consider explicitly requiring $x > 0$.

Listing 48: Require missing

```
127 z = int(log_2(uint128(x)));
```

3.49 CVF-49 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WadMath.sol

Recommendation This function would be much simpler if exp_2 would be able to deal with negative arguments.

Listing 49: Complicated code

```
184 function wadExp(uint x, uint y) internal pure returns (uint z) {
```

3.50 CVF-50 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WadMath.sol

Description b >>= 1 is redundant here as b is not used after this line.

Listing 50: Redundant code

```
294 b = b * b >> 127; if (b >= 0x100000000000000000000000000000000)
    ↪ {b >>= 1; l |= 0x10000000000000000;}
```

3.51 CVF-51 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Proxy.sol

Description The name of this contract is too generic. This is not just a proxy, but rather a USM proxy that allows using WETH instead of plain ether.
Recommendation So USM and WETH are worth mentioning in the name of this contract.

Listing 51: Bad naming

```
11 Proxy {
```

3.52 CVF-52 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Proxy.sol

Description It looks weird to use transferFrom rather than to transfer tokens from own address. Also, this code exploits known non-compliance of WETH with ERC-20 standard, as transferFrom own address doesn't use allowance. Consider using plain transfer here.

Listing 52: Complicated code

```
54 require(weth.transferFrom(address(this), to, ethOut), "WETH
   ↪ transfer fail");

83 require(weth.transferFrom(address(this), to, ethOut), "WETH
   ↪ transfer fail");
```

3.53 CVF-53 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Ownable.sol

Recommendation Events in Solidity are usually named via nouns, such as "OwnershipTransfer".

Listing 53: Bad naming

```
21 event OwnershipTransferred(address indexed previousOwner,
   ↪ address indexed newOwner);
```

3.54 CVF-54 Improper access specifiers

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Ownable.sol

Recommendation This function would not be necessary if the "_owner" storage variable would be public.

Listing 54: Improper access specifiers

```
34 function owner() public view returns (address) {
```

3.55 CVF-55 Redundant check

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Ownable.sol

Description This check is redundant, as it is still possible to transfer ownership to a dead address. If the idea was to prevent accidental ownership loss, then the common patter is to separate ownership transfer into two phases: offer ownership performed by the current owner and accept ownership performed by the new owner.

Listing 55: Redundant check

```
63 require(newOwner != address(0), "Ownable: new owner is the zero
   ↪ address");
```

3.56 CVF-56 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Ownable.sol

Description This event is logged even when the new owner is the same as the current one.

Listing 56: Redundant code

```
64 emit OwnershipTransferred(_owner, newOwner);
```

3.57 CVF-57 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** USMView.sol

Description The name "upOrDown" is odd as there could be other rounding modes, such as nearest or even rounding.
Recommendation Better name would be "rounding" or just "round".

Listing 57: Bad naming

```
24 function ethBuffer(WadMath.Round upOrDown) external view returns
   ↪  (int buffer) {

34 function ethToUsm(uint ethAmount, WadMath.Round upOrDown)
   ↪ external view returns (uint usmOut) {

44 function usmToEth(uint usmAmount, WadMath.Round upOrDown)
   ↪ external view returns (uint ethOut) {
```

3.58 CVF-58 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** USMView.sol

Description USM contract is called 5 times in the same function, where each call costs more than 1K gas. Moving this functionality into USM smart contract would save 4 out of 5 calls.

Listing 58: Impropper approach

```
72 (uint ethUsdPrice, ) = usm.latestPrice();
   uint ethPool = usm.ethPool();
   uint usmSupply = usm.totalSupply();
   uint oldTimeUnderwater = usm.timeSystemWentUnderwater();

77     (, usmSupply) = usm.checkIfUnderwater(usmSupply, ethPool,
         ↪ ethUsdPrice, oldTimeUnderwater, block.timestamp);

79 price = usm.fumPrice(side, ethUsdPrice, ethPool, usmSupply, usm.
     ↪ fumTotalSupply(), usm.buySellAdjustment());
```

3.59 CVF-59 Redundant check

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Address.sol

Description This check is redundant, as send will anyway fail in case the balance is insufficient.

Listing 59: Redundant check

```
26 require(address(this).balance >= amount, "Address: insufficient
     ↪ balance");
```

3.60 CVF-60 Improper interface placement

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundOpenOracle.sol

Recommendation This interface should be defined in its own file.

Listing 60: Improper interface placement

```
6 UniswapAnchoredView {
```

3.61 CVF-61 Comment missing

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** CompoundOpenOracle.sol

Description The number format of the returned value is unclear.
Recommendation Consider adding documentation comment.

Listing 61: Commnt missing

```
7 function price(string calldata symbol) external view returns (
    ↪ uint);
```

3.62 CVF-62 Improper access specifiers

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CompoundOpenOracle.sol

Description This constant doesn't need to be public.

Listing 62: Improper access specifiers

```
15 uint public constant COMPOUND_SCALE_FACTOR = 10 ** 12;  // Since
    ↪  Compound has 6 dec places, and latestPrice() needs 18
```

3.63 CVF-63 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** CompoundOpenOracle.sol

Recommendation 10e12 would be more readable.

Listing 63: Bad naming

```
15 uint public constant COMPOUND_SCALE_FACTOR = 10 ** 12;  // Since
    ↪  Compound has 6 dec places, and latestPrice() needs 18
```

3.64 CVF-64 Comment missing

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** CompoundOpenOracle.sol

Description The number format of this field is unclear.
Recommendation Consider adding a comment.

Listing 64: Comment missing

```
19 uint224 price;
```

3.65 CVF-65 Constant missing

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CompoundOpenOracle.sol

Recommendation The value "ETH" should be a named constant.

Listing 65: Constant missing

```
31 price = compoundUniswapAnchoredView.price("ETH") *
     ↪ COMPOUND_SCALE_FACTOR;
```

3.66 CVF-66 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundOpenOracle.sol

Description Both fields inside 'compoundStoredPrice' actually occupy the same storage slot, but this slot could be read two times.
Recommendation Consider reading the whole structure into memory at once to avoid double reading.

Listing 66: Improper approach

```
32 if (price != compoundStoredPrice.price) {

38     updateTime = compoundStoredPrice.updateTime;
```

3.67 CVF-67 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundOpenOracle.sol

Description This could be written as: compoundStoredPrice = TimedPrice ({ updateTime: uint32 (updateTime), price: uint224 (price) });

Listing 67: Improper approach

```
36 (compoundStoredPrice.updateTime, compoundStoredPrice.price) = (
     ↪ uint32(updateTime), uint224(price));
```

3.68 CVF-68 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundOpenOracle.sol

Recommendation This function doesn't have to update updateTime. It just needs to return it.

Listing 68: Improper approach

```
44 * retrieve the latest price, because as a view function, it (
     ↪ annoyingly) has no way to update updateTime, and (also
```

3.69 CVF-69 Check missing

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** CompoundOpenOracle.sol

Description This will return zero price and update time in case "refreshPrice" function wasn't called yet.
Recommendation Consider adding explicit check that updateTime > 0.

Listing 69: Check missing

```
48 (price, updateTime) = (compoundStoredPrice.price,
     ↪ compoundStoredPrice.updateTime);

52 (price, updateTime) = (compoundStoredPrice.price,
     ↪ compoundStoredPrice.updateTime);
```

3.70 CVF-70 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundOpenOracle.sol

Description This function does exactly the same as the "latestPrice" function. What is the reason to have two identical public functions in the same smart contract?

Listing 70: Complicated code

```
51 function latestCompoundPrice() public view returns (uint price,
     ↪ uint updateTime) {
```

3.71 CVF-71 Improper access specifier

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** ChainlinkOracle.sol

Description This constant doesn't need to be public. Remember, that public constants produce additional bytecode for getters and additional bytecode for dispatching, thus making.a contract more expensive to deploy and use.

Listing 71: Improper access specifier

```
12 uint public constant CHAINLINK_SCALE_FACTOR = 10 ** 10; // Since
   ↪ Chainlink has 8 dec places, and latestPrice() needs 18
```

3.72 CVF-72 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** ChainlinkOracle.sol

Recommendation 10e10 would be more readable.

Listing 72: Bad naming

```
12 uint public constant CHAINLINK_SCALE_FACTOR = 10 ** 10; // Since
   ↪ Chainlink has 8 dec places, and latestPrice() needs 18
```

3.73 CVF-73 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** ChainlinkOracle.sol

Description Usually, parameter names are prefixed with underscore ('_'), rather than have underscore as a suffix.

Listing 73: Bad naming

```
16 constructor(AggregatorV3Interface aggregator_)
```

3.74 CVF-74 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ChainlinkOracle.sol

Description This function does exactly the same as the "latestPrice" function. What is the reason to have two identical public functions in the same smart contract?

Listing 74: Complicated code

```
29 function latestChainlinkPrice() public view returns (uint price,
   ↪  uint updateTime) {
```

3.75 CVF-75 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ChainlinkOracle.sol

Recommendation The definition on the "rawPrice" variable could be moved inside the brackets like this: (, int rawPrice, , updateTime) = ...

Listing 75: Improper approach

```
30 int rawPrice;
(, rawPrice,, updateTime,) = chainlinkAggregator.latestRoundData
   ↪ ();
```

3.76 CVF-76 Underflow

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** ChainlinkOracle.sol

Description Underflow is possible when converting int to uint.
Recommendation Consider using safe cast library.

Listing 76: Underflow

```
32 price = uint(rawPrice) * CHAINLINK_SCALE_FACTOR; // TODO: Cast
   ↪ safely
```

3.77 CVF-77 File imported twice

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** FUM.sol

Description The same file is imported twice.

Listing 77: File imported twice

```
5  "./IUSM.sol";

9  "./IUSM.sol";
```

3.78 CVF-78 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FUM.sol

Description The contract stored USM address twice, once as the owner, and another time as the USM. This is confusing and suboptimal. Also, the word "stablecoin" here is confusing, as it is unclear that stablecoin=USM. Consider removing inheritance from "Ownable" and just put require (msg.sender == _usm) in the functions that now have onlyOwner modifiers. This would make code easier to read and more efficient.

Listing 78: Complicated code

```
16 * @notice This should be owned by the stablecoin.

18 FUM is ERC20Permit, WithOptOut, Ownable {
       IUSM public immutable usm;
```

3.79 CVF-79 Improper approach

- **Severity** Moderate
- **Status** Opened
- **Category** Unclear behavior
- **Source** FUM.sol

Description This is dangerous, as a user may want to set the minimum ether price in USD terms to be above $100000 thus putting non-zero value into the digits 8 to 11, but will end up with no minimum price set at all.

Listing 79: Improper approach

```
30 * If decimals 8 to 11 (included) of the amount of Ether received
       ↪ are `0000` then the next 7 will
   * be parsed as the minimum Ether price accepted, with 2 digits
       ↪ before and 5 digits after the comma.
```

3.80 CVF-80 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** FUM.sol

Description The price actually has 5 digits before and 2 digits after the point.

Listing 80: Improper comment

```
31 * be parsed as the minimum Ether price accepted, with 2 digits
   ↪ before and 5 digits after the comma.
```

3.81 CVF-81 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FUM.sol

Description Such may ways to defund makes the code overcomplicated and confuse users, while users still have many ways to loose tokens.
Recommendation Consider providing a single "right" way to defund.

Listing 81: Complicated code

```
43 if (recipient == address(this) || recipient == address(usm) ||
   ↪ recipient == address(0)) {
```

3.82 CVF-82 Ignored returned value

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** FUM.sol

Description The returned value is ignored here.

Listing 82: Ignored returned value

```
46 super._transfer(sender, recipient, amount);
```

3.83 CVF-83 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** WithOptOut.sol

Description The name is a bit confusing.
Recommendation Consider renaming.

Listing 83: Bad naming

```
5 contract WithOptOut {
```

3.84 CVF-84 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WithOptOut.sol

Description The contract tries to be generic by not specifying what exactly user are opting out of. However, a single contract may have several different things a user may opt out of, and this contract is not flexible enough to cover such case.

Listing 84: Improper approach

```
6 mapping(address => bool) public optedOut;  // true = address
    ↪ opted out of something
```

3.85 CVF-85 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** WithOptOut.sol

Description The deployer of the contract has a unique one-time ability to opt out others, while usually, one may only opt out himself. However, the number of addresses the deployer may opt out is limited by the block gas limit. This limit is hard to predict and thus seem weird...
Recommendation Consider implementing some way to opt out arbitrary large number of addresses at deployment time, for example by allowing to opt out more addresses in separate transactions and then seal the contract

Listing 85: Improper approach

```
8 constructor(address[] memory optedOut_) {
```

3.86 CVF-86 Not logged events

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** WithOptOut.sol

Recommendation These functions should probably log some events.

Listing 86: Not logged events

```
19 function optOut() public virtual {

23 function optBackIn() public virtual {
```

3.87 CVF-87 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Delegable.sol

Recommendation Just initialize the constant with keccak256 call. Solidity is smart enough to calculate teh hash at compile time.

Listing 87: Complicated code

```
10 // keccak256("Signature(address user,address delegate,uint256
   ↪ nonce,uint256 deadline)");
bytes32 public constant SIGNATURE_TYPEHASH = 0
   ↪ x0d077601844dd17f704bafff948229d27f33b57445915754dfe3d095fda2beb7
   ↪ ;
```

3.88 CVF-88 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Delegable.sol

Description The name "Signature" is confusing here, as it actually encodes arguments for "addDelegateBySignature" call. This could cause problems in case you will want to add another similar operation, such as "revokeDelegateBySignature".
Recommendation Consider renaming to "AddDelegate" or simply "Delegate".

Listing 88: Bad naming

```
10 // keccak256("Signature(address user,address delegate,uint256
   ↪ nonce,uint256 deadline)");
```

3.89 CVF-89 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Delegable.sol

Recommendation The word "nonce" is more common for this in the Ethereum world.

Listing 89: Bad naming

```
13 mapping(address => uint) public signatureCount;
```

3.90 CVF-90 Improper access specifier

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** OurUniswapV2TWAPOracle.sol

Description This constant doesn't need to be public.

Listing 90: Improper access specifier

```
23 uint public constant UNISWAP_MIN_TWAP_PERIOD = 10 minutes;
```

3.91 CVF-91 Improper access specifier

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** OurUniswapV2TWAPOracle.sol

Description This constant should not be public.

Listing 91: Improper access specifier

```
26 uint public constant UNISWAP_CUM_PRICE_SCALE_FACTOR = 2 ** 112;
```

3.92 CVF-92 ERC20 inconsistency

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** OurUniswapV2TWAPOracle.sol

Description In ERC20, "decimals" property is uint8, not uint.

Listing 92: ERC20 inconsistency

```
29 uint public immutable uniswapToken0Decimals;
30 uint public immutable uniswapToken1Decimals;

54 constructor(IUniswapV2Pair pair, uint token0Decimals, uint
     ↪ token1Decimals, bool tokensInReverseOrder) {
```

3.93 CVF-93 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** OurUniswapV2TWAPOracle.sol

Description Events in Solidity are usually named via nouns, such as just "TWAPPrice".

Listing 93: Bad naming

```
39 event TWAPPriceStored(uint32 timestamp, uint224 priceSeconds);
```

3.94 CVF-94 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** OurUniswapV2TWAPOracle.sol

Recommendation Decimals could be derived from Uniswap pair, no need to pass separately.

Listing 94: Complicated code

```
54 constructor(IUniswapV2Pair pair, uint token0Decimals, uint
   ↪ token1Decimals, bool tokensInReverseOrder) {
```

3.95 CVF-95 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** OurUniswapV2TWAPOracle.sol

Description This code doesn't support the case when aDecimals + 18 < bDecimals.

Listing 95: Improper approach

```
63 uniswapScaleFactor = 10 ** (aDecimals + 18 - bDecimals);
```

3.96 CVF-96 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** OurUniswapV2TWAPOracle.sol

Description This function does exactly the same as "latestPrice" function. What is the reason to have two identical public functions in the same smart contract?

Listing 96: Improper approach

```
84 function latestUniswapTWAPPrice() public view returns (uint
   ↪ price, uint updateTime) {
```

3.97 CVF-97 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** OurUniswapV2TWAPOracle.sol

Description Expressions uint32(timestamp) and uint224(cumPriceSeconds) were already calculated.
Recommendation Consider calculating them once and caching in local variables.

Listing 97: Complicated code

```
123 emit TWAPPriceStored(uint32(timestamp), uint224(cumPriceSeconds)
    ↪ ); // 4k gas, is it worth it?
```

3.98 CVF-98 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** OurUniswapV2TWAPOracle.sol

Description A call to this function is always preceded with a call to orderedStoredPrices, that reads and orders stored prices. If this function would receive ordered stored prices as a parameters, then it would not need to read them again, and the logic or this functions would be simpler and cheaper, as it will just need to check new newer price first and then the older price.

Listing 98: Improper approach

```
126 function storedPriceToCompareVs(uint newCumPriceSecondsTime,
    ↪ CumulativePrice storage newerStoredPrice)
```

3.99 CVF-99 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** OurUniswapV2TWAPOracle.sol

Description The name is confused as this function doesn't have an access to the new price, but only to the new timestamp.
Recommendation Consider renaming.

Listing 99: Bad naming

```
151 function areNewAndStoredPriceFarEnoughApart(uint newTimestamp,
    ↪ CumulativePrice storage storedPrice) internal view
```

3.100 CVF-100 Two versions of smart contract

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** OurUniswapV2TWAPOracle.sol

Description As 'uniswapTokensInReverseOrder' variable is immutable, only one of the branches is actually reachable in each particular instance of this smart contract.
Recommendation Consider having two versions of the smart contract: one for non-inverted pairs and another for inverted ones.

Listing 100: Two versions of smart contract

```
171 uint uniswapCumPrice = uniswapTokensInReverseOrder ?
```

3.101 CVF-101 Binary shift instead of division

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** OurUniswapV2TWAPOracle.sol

Recommendation Division here could be replaced with binary shift.

Listing 101: Binary shift instead of division

```
175 unchecked { cumPriceSeconds /= UNISWAP_CUM_PRICE_SCALE_FACTOR; }
```

3.102 CVF-102 Improper approach

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** OurUniswapV2TWAPOracle.sol

Description According to this comment: https://github.com/Uniswap/uniswap-v2-core/blob/v1.0.1/contracts/UniswapV2Pair.sol#L78 cumulative price may overflow, thus you should first calculate an unchecked difference between two raw cumulative prices, as returned by Uniswap, then divide the difference by timestamp difference, and only then apply scale factors. Otherwise, the oracle will stop working after cumulative price overflow.

Listing 102: Improper approach

```
188 price = (newCumPriceSeconds - oldCumPriceSeconds) / (
    ↪ newTimestamp - oldTimestamp);
```

3.103 CVF-103 Documentation comment missing

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Oracle.sol

Recommendation Consider using NatSpec comments for API documentation: https://docs.soliditylang.org/en/v0.8.0/natspec-format.html

Listing 103: Documentation comment missing

```
5 function latestPrice() public virtual view returns (uint price,
    ↪ uint updateTime);    // Prices WAD-scaled - 18 dec places

8     (price, updateTime) = latestPrice();    // Default
        ↪ implementation doesn't do any cacheing.  But override
        ↪ as needed
```

3.104 CVF-104 Typo

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Oracle.sol

Recommendation There s should be the "caching" instead of the cacheing.

Listing 104: Typo

```
8 (price, updateTime) = latestPrice();    // Default
    ↪ implementation doesn't do any cacheing.  But override as
    ↪ needed
```

3.105 CVF-105 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** MedianOracle.sol

Description "This oracle inherits three other oracles and allows obtaining four different prices: one per inherited oracle plus median. To achieve this, each inherited oracle exposes two identical functions that obtain latest price: one with standard name ""latestPrice"" and another with oracle specific name, such as ""latestUniswapTWAPPrice"". This design looks quite odd, as inherited oracles know that they will be inherited. More elegant design would be for basic oracles to just expose ""latestPrice"" functions, and median oracle to define separate public functions to obtain prices from inherited oracles like this: function latestUniswapTWAPPrice() public view returns (uint price, uint updateTime) { return OurUniswapV2TWAPOracle.latestPrice(); }"

Listing 105: Improper approach

```
8 MedianOracle is ChainlinkOracle, CompoundOpenOracle,
    ↪ OurUniswapV2TWAPOracle {
```

3.106 CVF-106 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** MedianOracle.sol

Recommendation Decimals parameters could be derived from the Uniswap pair. No need to pass them separately.

Listing 106: Complicated code

```
13 IUniswapV2Pair uniswapPair, uint uniswapToken0Decimals, uint
    ↪ uniswapToken1Decimals, bool uniswapTokensInReverseOrder
```

3.107 CVF-107 Dublicated code

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** MedianOracle.sol

Description This function does exactly the same as the "latestPrice" function. What is the reason to have two identical public functions in the same smart contract?

Listing 107: Dublicated code

```
25 function latestMedianOraclePrice() public view returns (uint
    ↪ price, uint updateTime) {
```

3.108 CVF-108 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** MedianOracle.sol

Description This saves one internal call, which should be insignificant from the gas point of view.

Listing 108: Improper approach

```
39 // this saves significant gas:
```

3.109 CVF-109 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** MedianOracle.sol

Description This code duplicates the code in the "latestMedianOraclePrice".
Recommendation Consider extracting to a utility function.

Listing 109: Complicated code

```
43 uint index = medianIndex(chainlinkPrice, compoundPrice,
     ↪ uniswapPrice);
(price, updateTime) = (index == 0 ? (chainlinkPrice,
     ↪ chainlinkUpdateTime) :
(index == 1 ? (compoundPrice, compoundUpdateTime) :
(uniswapPrice, uniswapUpdateTime)));
```

3.110 CVF-110 Improper access specifiers

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** MedianOracle.sol

Recommendation This function should not be public.

Listing 110: Improper access specifiers

```
53 function medianIndex(uint a, uint b, uint c)
       public pure returns (uint index)
```

3.111 CVF-111 Improper approach

- **Severity** Major
- **Status** Opened
- **Category** Unclear behavior
- **Source** MinOut.sol

Description This allows a user to specify the maximum token price between 0.00001 ETH and 100 ETH. Such range doesn't seem sufficient taking into account ETH price volatility and even greater possible volatility of FUM price. Also, the precision of the price cat degrated significantly when approaching the uppoer limit of 100 ETH per token.

Listing 111: Improper approach

```
7  if (minPrice != 0 && minPrice < 10000000) {
       minTokenOut = ethIn * minPrice / 100;
```

3.112 CVF-112 Magic number

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** MinOut.sol

Description The values 100000000000, 10000000, and 100 should be made a named constants.

Listing 112: Magic number

```
6  uint minPrice = ethIn % 100000000000;
   if (minPrice != 0 && minPrice < 10000000) {
       minTokenOut = ethIn * minPrice / 100;

13 uint maxPrice = tokenIn % 100000000000;
   if (maxPrice != 0 && maxPrice < 10000000) {
       minEthOut = tokenIn * 100 / maxPrice;
```

3.113 CVF-113 Improper approach

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** MinOut.sol

Description In case minPrice $\geq$ 10000000, its value is silently ignored. This is very dangerous, as a user, who wanted to specify very little price cap could end up with no cap set at all.

Listing 113: Improper approach

```
9  }
```

3.114 CVF-114 Improper approach

- **Severity** Major
- **Status** Opened
- **Category** Unclear behavior
- **Source** MinOut.sol

Description This allows a user to specify the minimum token price between 0.00001 ETH and 100 ETH. Such range doesn't seem sufficient taking into account ETH price volatility and even greater possible volatility of FUM price. Also, the precision of the lower price cap depgrades significantly when approaching the maximum the uppoer limit of 100 ETH per token.

Listing 114: Improper approach

```
14 if (maxPrice != 0 && maxPrice < 10000000) {
       minEthOut = tokenIn * 100 / maxPrice;
```

3.115 CVF-115 Improper approach

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** MinOut.sol

Description In case maxPrice $\geq$ 10000000, its value is silently ignored. This is very dangerous, as a user, who wanted to specify very little lower price cap could end up with no cap set at all.

Listing 115: Improper approach

```
16 }
```