Report

v. 1.0

Customer

Herodotus

Circuit and Smart Contract Audit

Herodotus

7th September 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	06.09.23	A. Zveryanskaya	Initial Draft
0.2	06.09.23	A. Zveryanskaya	Minor revision
1.0	06.09.23	A. Zveryanskaya	Release
1.1	07.09.23	D. Khovratovich	Minor revision

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) 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.

Herodotus aims to provide smart contracts with synchronous access to on-chain data coming from other Ethereum layers.

3 Project scope

We were asked to review:

- Original Code
- Solidity Fixes
- Rust Fixes

Files:

/			
	AggregatorsFactory.sol	SharpFacts Aggregator.sol	
libs/			
	block_header.cairo	mmr.cairo	utils.cairo
single_chunk_processor/			
	chunk_processor.cairo		
lib/			
	Uint256Splitter.sol		
interfaces/			
	IFactsRegistry.sol		

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code 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 as well as 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 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 if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure 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.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 1 critical, 9 major, and a few less important issues. All identified Critical and major issues have been fixed or resolved.

Issues	Info	Fixed
Severity		
Critical	0	1
Major	0	9
Moderate	3	6
Minor	13	22

Fixed 38 out of 54 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** utils.cairo

Description These checks allow negative values for b0, b1, ..., b7, which could be abused. For example: b0, b1, ..., b5 = 0 b6 = 1 b7 = -256 In such a case: 255 - b7 = 255 - (-256) = 513 < 2^128 (range check passed) word = 1 * 256 + (-256) = 0 but the returned value will be: 1 * 256^6 + (-256) * 256^7 != 256^6 - 256^8 != 0

Recommendation Consider adding range checks to disallow negative values for b0, ..., b7.

```
101 assert [range_check_ptr] = 255 - b0;
    assert [range_check_ptr + 1] = 255 - b1;
    assert [range_check_ptr + 2] = 255 - b2;
    assert [range_check_ptr + 3] = 255 - b3;
    assert [range_check_ptr + 4] = 255 - b4;
    assert [range_check_ptr + 5] = 255 - b5;
    assert [range_check_ptr + 6] = 255 - b6;
    assert [range_check_ptr + 7] = 255 - b7;
```

```
137 assert [range_check_ptr] = 255 - b0;
    assert [range_check_ptr + 1] = 255 - b1;
```

```
166 assert [range_check_ptr] = 255 - b0;
    assert [range_check_ptr + 1] = 255 - b1;
    assert [range_check_ptr + 2] = 255 - b2;
```

```
197 assert [range_check_ptr] = 255 - b0;
    assert [range_check_ptr + 1] = 255 - b1;
    assert [range_check_ptr + 2] = 255 - b2;
200 assert [range_check_ptr + 3] = 255 - b3;
```

```
230 assert [range_check_ptr] = 255 - b0;
    assert [range_check_ptr + 1] = 255 - b1;
    assert [range_check_ptr + 2] = 255 - b2;
    assert [range_check_ptr + 3] = 255 - b3;
    assert [range_check_ptr + 4] = 255 - b4;
```

(265, 302)

7 Major Issues

CVF-3. FIXED

- **Category** Unclear behavior
- **Source** utils.cairo

Description This function doesn't ensure that "y" is a power of two.

Recommendation Consider either implementing such a check or clearly stating that the caller is responsible for for, otherwise the function behavior is undefined.

Client Comment *It was already stated in the function's comment, but we made it more explicit.*

```
17 func bitwise_divmod{bitwise_ptr: BitwiseBuiltin*}(x: felt, y: felt)
      ↪ -> (q: felt, r: felt) {
```

CVF-4. FIXED

- **Category** Unclear behavior
- **Source** utils.cairo

Description This implicitly requires "value" to fit into 128 + 32 = 160 bits.

Recommendation Consider either modifying the code to support arbitrary argument values, or clearly stating this function limitation.

Client Comment *Added comment.*

```
34 let q = [range_check_ptr + 1];
```

CVF-7. FIXED

- **Category** Documentation
- **Source** mmr.cairo

Recommendation It should be checked that bit_length is from 1 to 127.

Client Comment *We were aware a hack is possible by accessing any ap value with a cutsom index (see tests/../dw_hack_test.cairo. However, to make the verification pass one should find two consecutives values of 2^b and 2^(b-1) somewhere in ap and that's why we wrote them in reverse order in the memory. Nevertheles we added the check from 0 to 127. The 0 case is failing the assertions.*

```
36 tempvar N = pow2_array[bit_length];
   tempvar n = pow2_array[bit_length - 1];
```

```
73 let N = pow2_array[bit_length];
   let n = pow2_array[bit_length - 1];
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** mmr.cairo

Description Computing a position height here is waste of resources is the height is already known to the caller.

Recommendation Consider passing the height as an argument.

```
129 let height = compute_height_pre_alloc_pow2(mmr_pos);
```

```
164 let height = compute_height_pre_alloc_pow2(left_child);
```

CVF-9. FIXED

- **Category** Overflow/Underflow
- **Source** mmr.cairo

Description Underflow is possible here.

Client Comment *This should never happen given the added initial check on the previous mmr_size from the main program: mmr_pos is always >=1. Added documentation.*

```
130 let right_sibling = mmr_pos + pow2_array[height + 1] - 1;
```

CVF-10. FIXED

- **Category** Unclear behavior
- **Source** mmr.cairo

Description The behaviour is undefined if mmr_len is not range checked.

Recommendation Consider adding a full check or state explicitly in the requirements that the caller must do so.

Client Comment *Added initial check to main program with assert_mmr_size_is_valid and updated docs*

```
147 func left_child_jump_until_inside_mmr{range_check_ptr, pow2_array:
    ↪ felt*, mmr_len}(
```

```
156         assert [range_check_ptr] = mmr_len - left_child;
```

```
161         assert [range_check_ptr] = left_child - mmr_len - 1;
```

CVF-11 FIXED

- **Category** Unclear behavior
- **Source** mmr.cairo

Description The behaviour is undefined if ‘mmr_offset‘ is not range checked.

Recommendation Consider adding a full check or state explicitly in the requirements that the caller must do so.

Client Comment *Added docs.*

```
190 mmr_offset: felt,
```

CVF-12. FIXED

- **Category** Unclear behavior
- **Source** mmr.cairo

Description The behaviour is undefined if ‘position‘ is not range checked.

Recommendation Consider adding a full check or state explicitly in the requirements that the caller must do so.

Client Comment *Added docs.*

```
193 }(position: felt) -> (peak_poseidon: felt, peak_keccak: Uint256) {
```

```
201         assert [range_check_ptr] = position - mmr_offset - 1;
```

```
203         let peak_poseidon = mmr_array_poseidon[position - mmr_offset
               ↪  - 1];
            let peak_keccak = mmr_array_keccak[position - mmr_offset -
               ↪ 1];
```

CVF-13. FIXED

- **Category** Unclear behavior
- **Source** mmr.cairo

Description The behaviour is undefined if ‘peaks_len‘ is not range checked.

Recommendation Consider adding a full check or state explicitly in the requirements that the caller must do so.

Client Comment *Added documentation.*

```
291 }(peaks_len: felt) -> (peaks_poseidon: felt*, peaks_keccak: Uint256
       ↪ *) {
```

```
295     get_peaks_from_positions_inner(peaks_poseidon, peaks_keccak,
           ↪ peaks_len - 1);
```

CVF-16. FIXED

- **Category** Procedural
- **Source** SharpFactsAggregator.sol

Description Several different errors are used to signal basically the same fact, that an output is not a valid continuation for the current state.

Recommendation Consider using a single error.

Client Comment *Will apply this feedback.*

```
360 revert AggregationPoseidonRootMismatch();
```

```
364 revert AggregationKeccakRootMismatch();
```

```
368 revert AggregationSizeMismatch();
```

```
377     revert AggregationErrorParentHashMismatch();
```

```
383     revert AggregationErrorParentHashMismatch();
```

8 Moderate Issues

CVF-5. INFO

- **Category** Suboptimal
- **Source** utils.cairo

Description These two functions basically do the same.

Recommendation Consider removing one of these functions.

Client Comment *CVF34 related. No changes made.*

```
57 func word_reverse_endian_64{bitwise_ptr: BitwiseBuiltin*}(word: felt
   ↪ ) -> (res: felt) {
```

```
82 func word_reverse_endian_64_RC{range_check_ptr}(word: felt) -> felt
   ↪ {
```

CVF-6. INFO

- **Category** Procedural
- **Source** mmr.cairo

Recommendation A more traditional notation is to use zero-based indexes.

Client Comment *We use 1-based index to benefit from the fact that the leftmost nodes at each height is of the form 2^h-1 (all ones), as it is commonly the case in several implementations. No changes made.*

```
14 // 2         7
   //         /   \
   // 1     3     6
   //      / \   / \
   // 0   1   2 4   5
```

CVF-14. INFO

- **Category** Suboptimal
- **Source** block_header.cairo

Recommendation A more efficient “bitwise_divmod” function could be used here.

Client Comment *We are satistsfied with the current approach. See CVF-34.*

```
142 let (first_byte, remaining_7) = felt_divmod(rlp_difficulty, 2 ** 56)
      ↪ ;
```

```
160     let (first_byte, remainder) = felt_divmod(
```

```
181     let (first_byte, remaining_7) = felt_divmod(block_number_item, 2
          ↪  ** 56);
```

```
215 let (number_of_exact_8bytes_chunks, number_of_bytes_in_last_chunk) =
      ↪  felt_divmod(n_bytes, 8);
    let (_, rem) = felt_divmod(seed, 2);
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** block_header.cairo

Recommendation Consider using a binary decision tree instead of a linear one.

Client Comment *We are satistsfied with the current approach.*

```
238 if (number_of_bytes_in_last_chunk == 1) {
```

```
245 if (number_of_bytes_in_last_chunk == 2) {
```

```
253 if (number_of_bytes_in_last_chunk == 3) {
```

```
261 if (number_of_bytes_in_last_chunk == 4) {
```

```
269 if (number_of_bytes_in_last_chunk == 5) {
```

```
277 if (number_of_bytes_in_last_chunk == 6) {
```

```
285 if (number_of_bytes_in_last_chunk == 7) {
```

CVF-17. FIXED

- **Category** Unclear behavior
- **Source** mmr.cairo

Description This way of hashing is collision prone.

Recommendation Consider prefixing the input length to guarantee that different-length MMRs give different hashes.

```
339 // Hashes the peaks of both MMRs together by computing H(peak1, H(
    ↪ peak2, H(peak3, ...))).
```

CVF-18. INFO

- **Category** Overflow/Underflow
- **Source** block_header.cairo

Description Underflow is possible when calculating the “pow2_array” index.

Recommendation Consider adding appropriate range checks.

Client Comment *Given that from the merge the difficulty is 0, this should not happen unless the block number is approximately 2^88 times larger than it is today. No changes made.*

```
174 remainder, pow2_array[56 + (7 - difficulty_offset) * 8 -
    ↪ block_number_offset * 8]
```

CVF-19. FIXED

- **Category** Suboptimal
- **Source** Uint256Splitter.sol

Description This formula produces incorrect result in case “lower” exceeds 128 bits.

Recommendation Consider doing on of the following: 1. Add a “require” statement to ensure “lower” does fit into 128 bits. 2. Change the type of “lower” to “uint128” and use bitwise “AND” to clear the higher 128 bits (Solidity allows junk in unused bits of types shorter than 256 bits). 3. Clearly state in a comment that the caller is responsible for “lower” to always fit into 128 bits.

```
24 a := or(shl(128, upper), lower)
```

CVF-20. FIXED

- **Category** Suboptimal
- **Source** AggregatorsFactory.sol

Description An upgrade proposal cannot be revoked, and it is possible to executed arbitrary non-executed proposals from the past. This means that if a broken proposal was every proposed, it will always propose a thread for the protocol.

Recommendation Consider either implementing an ability to revoke a proposed upgrade, or allowing only the latest upgrade proposal to be executed.

```
155 require(updateId <= upgradesCount, "Invalid updateId");
```

CVF-21. INFO

- **Category** Procedural
- **Source** SharpFactsAggregator.sol

Recommendation The fact should also hash the StarkNet chain ID in order to prevent replay attacks on different networks.

Client Comment *Interesting, we can't just change the fact.*

```
341 bytes32 fact = keccak256(abi.encode(PROGRAM_HASH, outputHash));
```

```
434 bytes32 fact = keccak256(abi.encode(PROGRAM_HASH, outputHash));
```

9 Minor Issues

CVF-2. INFO

- **Category** Unclear behavior
- **Source** chunk_processor.cairo

Description The block headers are not verified to be in the Keccak input format (64 bits per felt). The inner keccak code does make the range checks, but the documentation of the builtin does not tell about it but rather expects a properly formed input. This means that the checks may be removed in the future.

Recommendation Consider doing your own check.

Client Comment *Disagree → it would lead to 2x range checks, we are protected as a change in keccak core lib (extremely unlikely) would lead to a different program hash. We mentioned the lack of documentation to Starkware.*

```
301 let (block_headers_array: felt**, bytes_len_array: felt*) =
       ↪ read_block_headers();
```

CVF-22. FIXED

- **Category** Bad naming
- **Source** chunk_processor.cairo

Recommendation The argument name is confusing. It is actually not a parent hash, but rather the expected block hash to verify.

```
45 // - parent_hash: Uint256 - parent hash of block header i+1 (little
       ↪ endian)
```

CVF-23. FIXED

- **Category** Procedural
- **Source** chunk_processor.cairo

Recommendation Consider indicating the range of the parameter for which the function executes correctly, and assert in the witness code.

Client Comment *Added comments.*

```
59 }(index: felt, parent_hash: Uint256) -> (
```

```
131 }(index: felt) {
```

```
185 }(height: felt) {
```

```
499     index: felt, peaks_positions: felt*, peaks_values_poseidon: felt
        ↪ *, peaks_values_keccak: Uint256*
```

CVF-24. FIXED

- **Category** Suboptimal
- **Source** chunk_processor.cairo

Recommendation This could be simplified as: right_pos = next_pos - 1

Client Comment *Similarly, in Line 192 (next_pos_is_parent != 0) could be simplified to (height_next_pos==height+1) by construction of the MMR, avoiding one range check.*

```
202 local right_pos = left_pos + pow2_array[height + 1] - 1;
```

CVF-25. FIXED

- **Category** Procedural
- **Source** utils.cairo

Recommendation Constants are usually spelled in UPPER_CASE.

```
6 const div_32 = 2 ** 32;
const div_32_minus_1 = div_32 - 1;
```

CVF-26. FIXED

- **Category** Documentation
- **Source** utils.cairo

Recommendation Should be ">=".

```
50 // The result will not make sense if word > 2^64.
```

CVF-27. INFO

- **Category** Suboptimal
- **Source** utils.cairo

Recommendation Consider also implementing 32- and 16-bit version of this function.

Client Comment *See CVF-34. No changes made.*

```
57 func word_reverse_endian_64{bitwise_ptr: BitwiseBuiltin*}(word: felt
   ↪ ) -> (res: felt) {
```

CVF-28. FIXED

- **Category** Unclear behavior
- **Source** utils.cairo

Recommendation The mask should be: 0×00ff00ff00ff00ff

Client Comment *The current version was working but indeed those maks were for a 128 bit number.*

```
60 assert bitwise_ptr[0].y = 0x00ff00ff00ff00ff00ff00ff00ff00ff;
```

CVF-29. FIXED

- **Category** Unclear behavior
- **Source** utils.cairo

Recommendation The mask should be: 0×0000ffff0000ffff00

Client Comment *The current version was working but indeed those maks were for a 128 bit number.*

```
64 assert bitwise_ptr[1].y = 0x00ffff0000ffff0000ffff0000ffff00;
```

CVF-30. FIXED

- **Category** Unclear behavior
- **Source** utils.cairo

Recommendation The mask should be: 0×00000000ffffffff000000

Client Comment *The current version was working but indeed those maks were for a 128 bit number.*

```
68 assert bitwise_ptr[2].y = 0x00ffffffff00000000ffffffff000000;
```

CVF-31. FIXED

- **Category** Suboptimal
- **Source** mmr.cairo

Description The expression “position - mmr_offset -1” is calculated several times.

Recommendation Consider calculating once and reusing.

```
201 assert [range_check_ptr] = position - mmr_offset - 1;
```

```
203 let peak_poseidon = mmr_array_poseidon[position - mmr_offset - 1];
    let peak_keccak = mmr_array_keccak[position - mmr_offset - 1];
```

CVF-32. INFO

- **Category** Suboptimal
- **Source** block_header.cairo

Recommendation Consider writing this as a strong requirement and assert in the python code.

Client Comment *We are satistsfied with the current documentation.*

```
51 // - rlp: felt* - A pointer to the array of felts, each segmenting
      ↪ the block header into 8-byte little endian chunks.
```

```
104 // Assumes the block header is correctly RLP encoded with values as
       ↪ 64-bit big endian.
```

CVF-33. FIXED

- **Category** Suboptimal
- **Source** block_header.cairo

Description The maximum difficulty recorded may change in the future. Relying on it seems weird.

Recommendation Consider explicitly asserting that difficulty offset doesn't exceed 7.

Client Comment *Since the merge and Proof of Stake, difficulty field is always 0. Yet we added this check since we only call this function once per job.*

```
144 // MAX Difficulty recorded is 15911382925018176 so max difficulty
       ↪ offset is 7, and will alway fit in the remaining_7.
```

CVF-34. INFO

- **Category** Suboptimal
- **Source** block_header.cairo

Description Bitwise version uses three bitwise operations, while range check version uses 8 range checks. Alternating them doesn't seem fair.

Client Comment *We are constrained by SHARP resources per job, and builtins invocations can be combined but none of them can exceed its individual limit. After some tuning we decided to split bitwise and RC in a 60/40% ratio to process the maximum number of blocks possible in one job. We use divmod(seed, 10) and then process depending on the range of the remainder to make it 60/40.*

```
204 // - seed: felt - Determines resource allocation between range check
       ↪  and bitwise operations.
    //                If even, range check is used, else bitwise.
```

CVF-35. INFO

- **Category** Suboptimal
- **Source** block_header.cairo

Recommendation It would probably be more efficient to always reverse all 8 bytes and then select proper number of bytes from the result.

Client Comment *We are satistsfied with the current approach.*

```
246 let last_reversed_chunk: felt = word_reverse_endian_16_RC(
```

```
254 let last_reversed_chunk: felt = word_reverse_endian_24_RC(
```

```
262 let last_reversed_chunk: felt = word_reverse_endian_32_RC(
```

```
270 let last_reversed_chunk: felt = word_reverse_endian_40_RC(
```

```
278 let last_reversed_chunk: felt = word_reverse_endian_48_RC(
```

```
286 let last_reversed_chunk: felt = word_reverse_endian_56_RC(
```

CVF-36. FIXED

- **Category** Readability
- **Source** Uint256Splitter.sol

Recommendation This value could be specified as “type(uint128).max”.

```
5 uint256 constant _MASK = (1 << 128) - 1;
```

CVF-37. FIXED

- **Category** Procedural
- **Source** Uint256Splitter.sol

Description The names and types of the returned values differ between the comment and the code.

Recommendation Consider making them consistent.

```
7  /// @notice Splits a uint256 into two uint128s (low, high).
```

```
11 ) internal pure returns (uint256 lower, uint256 upper) {
```

CVF-38. FIXED

- **Category** Procedural
- **Source** Uint256Splitter.sol

Description The argument types in the comment and in the code are different.

Recommendation Consider making them consistent.

```
16 /// @param lower The lower uint128.
   /// @param upper The upper uint128.
```

```
19     uint256 lower,
20     uint256 upper
```

CVF-39. FIXED

- **Category** Bad datatype
- **Source** AggregatorsFactory.sol

Recommendation The type of this variable should be “SharpFactsAggregator” or an interface extracted from it.

```
15 address public template;
```

CVF-40. FIXED

- **Category** Bad datatype
- **Source** AggregatorsFactory.sol

Recommendation The type of this field should be “SharpFactsAggregator” or an interface extracted from it.

```
20 address newTemplate;
```

CVF-41. FIXED

- **Category** Bad datatype
- **Source** AggregatorsFactory.sol

Recommendation The value type should be “SharpFactsAggregator” or an interface extracted from it.

```
36 mapping(uint256 => address) public aggregatorsById;
```

CVF-42. INFO

- **Category** Procedural
- **Source** AggregatorsFactory.sol

Recommendation Events parameters should be indexed.

Client Comment *This wouldn’t add much value to our usecase and would increase the gas cost.*

```
51 event UpgradeProposal(address newTemplate);
   event Upgrade(address oldTemplate, address newTemplate);
   event AggregatorCreation(address aggregator, uint256 aggregatorId);
```

CVF-43. INFO

- **Category** Suboptimal
- **Source** AggregatorsFactory.sol

Recommendation The “oldTermplate“ parameter is redundant, as its value could be derived from the previous events.

Client Comment *We want to make it explicit at first glance.*

```
52 event Upgrade(address oldTemplate, address newTemplate);
```

CVF-44. INFO

- **Category** Bad datatype
- **Source** AggregatorsFactory.sol

Recommendation The type of the address parameters for these events should be “SharpFactsAggregator” or an interface extracted from it.

Client Comment *We prefer dealing with the address type as it’s less opionionated.*

```
51 event UpgradeProposal(address newTemplate);
   event Upgrade(address oldTemplate, address newTemplate);
   event AggregatorCreation(address aggregator, uint256 aggregatorId);
```

CVF-45. INFO

- **Category** Bad datatype
- **Source** AggregatorsFactory.sol

Recommendation The argument type should be “SharpFactsAggregator” or an interface extracted from it.

Client Comment *Same as above.*

```
141 function proposeUpgrade(address newTemplate) external onlyOperator {
```

CVF-46. INFO

- **Category** Bad datatype
- **Source** SharpFactsAggregator.sol

Recommendation The type of this constant should be “IFactsRegistry”.

Client Comment *Same as above.*

```
32 address public constant FACTS_REGISTRY =
```

CVF-47. INFO

- **Category** Procedural
- **Source** SharpFactsAggregator.sol

Description Hardcoding mainnet addresses is a bad practice, as it makes it harder to test code.

Recommendation Consider turning this constant into an immutable variable and passing its value as a constructor argument.

Client Comment *We agree but are okay with testing more on this matter.*

```
32 address public constant FACTS_REGISTRY =
       0xAB43bA48c9edF4C2C4bB01237348D1D7B28ef168; // Goërli
```

CVF-48. FIXED

- **Category** Suboptimal
- **Source** SharpFactsAggregator.sol

Recommendation These errors could be made more useful by adding some parameters into them.

```
91 error NotEnoughBlockConfirmations();
   error TooManyBlocksConfirmations();
   error NotEnoughJobs();
   error UnknownParentHash();
   error AggregationPoseidonRootMismatch();
   error AggregationKeccakRootMismatch();
   error AggregationSizeMismatch();
   error AggregationErrorParentHashMismatch();
   error AggregationBlockMismatch();
```

```
101 error InvalidFact();
```

CVF-49. FIXED

- **Category** Unclear behavior
- **Source** SharpFactsAggregator.sol

Recommendation This function should emit some event.

```
164 function setOperatorRequired(
```

CVF-50. FIXED

- **Category** Bad datatype
- **Source** SharpFactsAggregator.sol

Recommendation The values “20” and “255” should be named constants.

```
177 if (blocksConfirmations < 20) {
```

```
183 if (blocksConfirmations > 255) {
```

CVF-51. INFO

- **Category** Suboptimal
- **Source** SharpFactsAggregator.sol

Recommendation There is a trick how to range check a value in less gas: unchecked { if (blocksConfirmations - 20 > 235) revert (); // Relies on underflow behavior }

Client Comment *Makes the code less readable and the cost saving is insignificant.*

177
```
if (blocksConfirmations < 20) {
```

183
```
if (blocksConfirmations > 255) {
```

CVF-52. INFO

- **Category** Suboptimal
- **Source** SharpFactsAggregator.sol

Description The function allows registering the same block multiple times, emitting several events with the same parameters.

Recommendation Consider explicitly disallowing registering a block that is already registered.

Client Comment *If there is reorg and we want to overwrite the reorged blockhash we won't be able to do so.*

209
```
emit NewRangeRegistered(targetBlock, targetBlockParentHash);
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** SharpFactsAggregator.sol

Description This check is redundant, as in case there is only one output, the loop inside will execute zero times.

Recommendation Consider removing this check.

255
```
if (outputs.length > 1) {
```

CVF-54. FIXED

- **Category** Suboptimal
- **Source** SharpFactsAggregator.sol

Recommendation This variable is redundant. Just return the expression value.

```
436 bool isValidFact = IFactsRegistry(FACTS_REGISTRY).isValid(fact);
```

CVF-55. FIXED

- **Category** Suboptimal
- **Source** SharpFactsAggregator.sol

Recommendation This function is redundant as the “aggregatorState“ variable is already public.

```
441 function getAggregatorState()
```

ABDK
Consulting

About us

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.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting