ABDK CONSULTING

SMART CONTRACT
AUDIT

EdgeSwap

Solidity

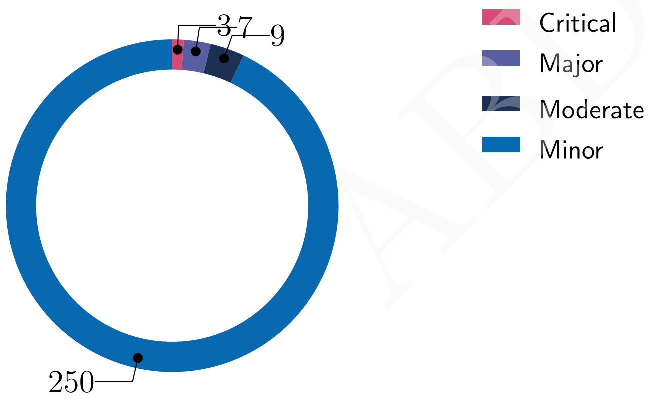
abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich

11th November 2021

We've been asked to review the 31 files in a Github repo. We found 3 critical, 7 major, and a few less important issues. All critical and major issues were fixed.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Bad naming	Opened
CVF-2	Minor	Suboptimal	Opened
CVF-3	Minor	Procedural	Opened
CVF-4	Minor	Bad datatype	Opened
CVF-5	Minor	Suboptimal	Opened
CVF-6	Minor	Suboptimal	Opened
CVF-7	Minor	Bad datatype	Opened
CVF-8	Minor	Flaw	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Minor	Overflow/Underflow	Opened
CVF-11	Minor	Suboptimal	Opened
CVF-12	Minor	Readability	Opened
CVF-13	Minor	Suboptimal	Opened
CVF-14	Minor	Suboptimal	Opened
CVF-15	Minor	Suboptimal	Opened
CVF-16	Minor	Suboptimal	Opened
CVF-17	Minor	Readability	Opened
CVF-18	Minor	Documentation	Opened
CVF-19	Minor	Procedural	Opened
CVF-20	Minor	Bad naming	Opened
CVF-21	Minor	Bad datatype	Opened
CVF-22	Critical	Flaw	Fixed
CVF-23	Minor	Suboptimal	Opened
CVF-24	Minor	Suboptimal	Opened
CVF-25	Minor	Procedural	Opened
CVF-26	Minor	Suboptimal	Opened
CVF-27	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Suboptimal	Opened
CVF-30	Minor	Procedural	Opened
CVF-31	Minor	Suboptimal	Opened
CVF-32	Minor	Procedural	Opened
CVF-33	Minor	Suboptimal	Opened
CVF-34	Minor	Flaw	Opened
CVF-35	Minor	Bad naming	Opened
CVF-36	Minor	Bad datatype	Opened
CVF-37	Minor	Procedural	Opened
CVF-38	Minor	Bad naming	Opened
CVF-39	Minor	Suboptimal	Opened
CVF-40	Minor	Bad datatype	Opened
CVF-41	Minor	Procedural	Opened
CVF-42	Minor	Unclear behavior	Opened
CVF-43	Minor	Unclear behavior	Opened
CVF-44	Minor	Procedural	Opened
CVF-45	Minor	Documentation	Opened
CVF-46	Minor	Documentation	Opened
CVF-47	Minor	Suboptimal	Opened
CVF-48	Minor	Documentation	Opened
CVF-49	Minor	Bad naming	Opened
CVF-50	Minor	Suboptimal	Opened
CVF-51	Minor	Bad datatype	Opened
CVF-52	Major	Flaw	Fixed
CVF-53	Moderate	Flaw	Opened
CVF-54	Minor	Suboptimal	Opened
CVF-55	Minor	Suboptimal	Opened
CVF-56	Moderate	Flaw	Opened
CVF-57	Minor	Bad datatype	Opened

ID	Severity	Category	Status
CVF-58	Minor	Suboptimal	Opened
CVF-59	Minor	Documentation	Opened
CVF-60	Minor	Bad datatype	Opened
CVF-61	Minor	Bad datatype	Opened
CVF-62	Minor	Bad datatype	Opened
CVF-63	Minor	Bad datatype	Opened
CVF-64	Minor	Bad datatype	Opened
CVF-65	Critical	Flaw	Fixed
CVF-66	Minor	Overflow/Underflow	Opened
CVF-67	Minor	Procedural	Opened
CVF-68	Minor	Suboptimal	Opened
CVF-69	Minor	Suboptimal	Opened
CVF-70	Minor	Suboptimal	Opened
CVF-71	Minor	Bad naming	Opened
CVF-72	Moderate	Procedural	Opened
CVF-73	Minor	Bad datatype	Opened
CVF-74	Minor	Suboptimal	Opened
CVF-75	Minor	Suboptimal	Opened
CVF-76	Minor	Suboptimal	Opened
CVF-77	Minor	Bad naming	Opened
CVF-78	Minor	Bad naming	Opened
CVF-79	Minor	Suboptimal	Opened
CVF-80	Minor	Suboptimal	Opened
CVF-81	Minor	Bad datatype	Opened
CVF-82	Minor	Documentation	Opened
CVF-83	Minor	Suboptimal	Opened
CVF-84	Major	Flaw	Fixed
CVF-85	Minor	Suboptimal	Opened
CVF-86	Minor	Suboptimal	Opened
CVF-87	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-88	Minor	Procedural	Opened
CVF-89	Critical	Flaw	Fixed
CVF-90	Major	Suboptimal	Info
CVF-91	Minor	Suboptimal	Opened
CVF-92	Minor	Suboptimal	Opened
CVF-93	Minor	Suboptimal	Opened
CVF-94	Minor	Procedural	Opened
CVF-95	Minor	Procedural	Opened
CVF-96	Minor	Suboptimal	Opened
CVF-97	Minor	Suboptimal	Opened
CVF-98	Minor	Suboptimal	Opened
CVF-99	Minor	Suboptimal	Opened
CVF-100	Minor	Procedural	Opened
CVF-101	Moderate	Flaw	Opened
CVF-102	Minor	Bad datatype	Opened
CVF-103	Minor	Bad datatype	Opened
CVF-104	Minor	Bad datatype	Opened
CVF-105	Minor	Bad datatype	Opened
CVF-106	Minor	Bad datatype	Opened
CVF-107	Minor	Procedural	Opened
CVF-108	Minor	Suboptimal	Opened
CVF-109	Minor	Suboptimal	Opened
CVF-110	Major	Flaw	Fixed
CVF-111	Minor	Suboptimal	Opened
CVF-112	Minor	Bad datatype	Opened
CVF-113	Minor	Suboptimal	Opened
CVF-114	Minor	Suboptimal	Opened
CVF-115	Minor	Bad datatype	Opened
CVF-116	Minor	Bad datatype	Opened
CVF-117	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-118	Minor	Procedural	Opened
CVF-119	Minor	Suboptimal	Opened
CVF-120	Minor	Procedural	Opened
CVF-121	Minor	Bad datatype	Opened
CVF-122	Moderate	Flaw	Opened
CVF-123	Minor	Suboptimal	Opened
CVF-124	Minor	Bad datatype	Opened
CVF-125	Minor	Suboptimal	Opened
CVF-126	Minor	Suboptimal	Opened
CVF-127	Minor	Suboptimal	Opened
CVF-128	Minor	Documentation	Opened
CVF-129	Minor	Bad naming	Opened
CVF-130	Minor	Bad datatype	Opened
CVF-131	Minor	Bad datatype	Opened
CVF-132	Minor	Procedural	Opened
CVF-133	Minor	Procedural	Opened
CVF-134	Minor	Bad datatype	Opened
CVF-135	Minor	Bad naming	Opened
CVF-136	Minor	Unclear behavior	Opened
CVF-137	Minor	Bad naming	Opened
CVF-138	Minor	Bad naming	Opened
CVF-139	Minor	Suboptimal	Opened
CVF-140	Minor	Bad datatype	Opened
CVF-141	Minor	Suboptimal	Opened
CVF-142	Minor	Unclear behavior	Opened
CVF-143	Minor	Documentation	Opened
CVF-144	Minor	Procedural	Opened
CVF-145	Minor	Suboptimal	Opened
CVF-146	Minor	Procedural	Opened
CVF-147	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-148	Major	Flaw	Info
CVF-149	Minor	Suboptimal	Opened
CVF-150	Minor	Documentation	Opened
CVF-151	Minor	Procedural	Opened
CVF-152	Minor	Suboptimal	Opened
CVF-153	Minor	Readability	Opened
CVF-154	Minor	Overflow/Underflow	Opened
CVF-155	Minor	Procedural	Opened
CVF-156	Minor	Procedural	Opened
CVF-157	Moderate	Flaw	Opened
CVF-158	Minor	Suboptimal	Opened
CVF-159	Minor	Suboptimal	Opened
CVF-160	Moderate	Flaw	Opened
CVF-161	Minor	Suboptimal	Opened
CVF-162	Minor	Flaw	Opened
CVF-163	Minor	Bad datatype	Opened
CVF-164	Minor	Procedural	Opened
CVF-165	Minor	Procedural	Opened
CVF-166	Minor	Documentation	Opened
CVF-167	Minor	Procedural	Opened
CVF-168	Minor	Suboptimal	Opened
CVF-169	Minor	Suboptimal	Opened
CVF-170	Minor	Suboptimal	Opened
CVF-171	Minor	Procedural	Opened
CVF-172	Minor	Bad naming	Opened
CVF-173	Minor	Suboptimal	Opened
CVF-174	Minor	Procedural	Opened
CVF-175	Minor	Bad datatype	Opened
CVF-176	Minor	Suboptimal	Opened
CVF-177	Minor	Bad datatype	Opened

ID	Severity	Category	Status
CVF-178	Minor	Documentation	Opened
CVF-179	Minor	Documentation	Opened
CVF-180	Minor	Documentation	Opened
CVF-181	Minor	Readability	Opened
CVF-182	Minor	Procedural	Opened
CVF-183	Minor	Bad naming	Opened
CVF-184	Minor	Bad datatype	Opened
CVF-185	Minor	Bad datatype	Opened
CVF-186	Minor	Bad datatype	Opened
CVF-187	Minor	Bad naming	Opened
CVF-188	Minor	Bad datatype	Opened
CVF-189	Minor	Bad naming	Opened
CVF-190	Minor	Suboptimal	Opened
CVF-191	Minor	Procedural	Opened
CVF-192	Minor	Suboptimal	Opened
CVF-193	Minor	Procedural	Opened
CVF-194	Minor	Suboptimal	Opened
CVF-195	Minor	Bad datatype	Opened
CVF-196	Minor	Suboptimal	Opened
CVF-197	Minor	Procedural	Opened
CVF-198	Moderate	Bad naming	Opened
CVF-199	Minor	Bad datatype	Opened
CVF-200	Minor	Bad naming	Opened
CVF-201	Minor	Bad naming	Opened
CVF-202	Minor	Procedural	Opened
CVF-203	Minor	Unclear behavior	Opened
CVF-204	Minor	Suboptimal	Opened
CVF-205	Minor	Suboptimal	Opened
CVF-206	Minor	Procedural	Opened
CVF-207	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-208	Minor	Suboptimal	Opened
CVF-209	Minor	Bad datatype	Opened
CVF-210	Minor	Documentation	Opened
CVF-211	Minor	Bad naming	Opened
CVF-212	Minor	Bad datatype	Opened
CVF-213	Minor	Procedural	Opened
CVF-214	Minor	Documentation	Opened
CVF-215	Minor	Suboptimal	Opened
CVF-216	Minor	Suboptimal	Opened
CVF-217	Minor	Flaw	Opened
CVF-218	Minor	Suboptimal	Opened
CVF-219	Minor	Procedural	Opened
CVF-220	Minor	Procedural	Opened
CVF-221	Minor	Procedural	Opened
CVF-222	Minor	Procedural	Opened
CVF-223	Minor	Suboptimal	Opened
CVF-224	Minor	Suboptimal	Opened
CVF-225	Moderate	Procedural	Opened
CVF-226	Major	Flaw	Fixed
CVF-227	Minor	Suboptimal	Opened
CVF-228	Minor	Bad naming	Opened
CVF-229	Minor	Bad naming	Opened
CVF-230	Minor	Suboptimal	Opened
CVF-231	Minor	Suboptimal	Opened
CVF-232	Minor	Bad naming	Opened
CVF-233	Minor	Bad naming	Opened
CVF-234	Minor	Documentation	Opened
CVF-235	Minor	Procedural	Opened
CVF-236	Minor	Suboptimal	Opened
CVF-237	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-238	Minor	Suboptimal	Opened
CVF-239	Minor	Suboptimal	Opened
CVF-240	Minor	Bad datatype	Opened
CVF-241	Minor	Suboptimal	Opened
CVF-242	Minor	Bad datatype	Opened
CVF-243	Minor	Suboptimal	Opened
CVF-244	Minor	Procedural	Opened
CVF-245	Minor	Suboptimal	Opened
CVF-246	Minor	Suboptimal	Opened
CVF-247	Minor	Suboptimal	Opened
CVF-248	Minor	Suboptimal	Opened
CVF-249	Minor	Suboptimal	Opened
CVF-250	Minor	Suboptimal	Opened
CVF-251	Minor	Suboptimal	Opened
CVF-252	Minor	Procedural	Opened
CVF-253	Minor	Suboptimal	Opened
CVF-254	Minor	Suboptimal	Opened
CVF-255	Minor	Documentation	Opened
CVF-256	Minor	Bad naming	Opened
CVF-257	Minor	Suboptimal	Opened
CVF-258	Minor	Suboptimal	Opened
CVF-259	Minor	Suboptimal	Opened
CVF-260	Minor	Suboptimal	Opened
CVF-261	Minor	Suboptimal	Opened
CVF-262	Minor	Suboptimal	Opened
CVF-263	Minor	Bad datatype	Opened
CVF-264	Minor	Overflow/Underflow	Opened
CVF-265	Minor	Suboptimal	Opened
CVF-266	Major	Suboptimal	Info
CVF-267	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-268	Minor	Unclear behavior	Opened
CVF-269	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	November 10, 2021	D. Khovratovich	Initial Draft
0.2	November 11, 2021	D. Khovratovich	Minor revision
1.0	November 11, 2021	D. Khovratovich	Release
1.1	November 12, 2021	D. Khovratovich	Repository fixes
2.0	November 11, 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 have reviewed the contracts at audit tag:

- uniswap/interfaces/IUniswapV2Pair.sol
- uniswap/libraries/Math.sol
- uniswap/libraries/UQ112x112.sol
- uniswap/libraries/UniswapSafeMath.sol
- uniswap/UniswapV2ERC20.sol
- uniswap/UniswapV2Factory.sol
- uniswap/UniswapV2Pair.sol
- Bytes.sol
- Config.sol
- DeployFactory.sol
- Events.sol
- Governance.sol
- IERC20.sol
- Operations.sol
- Ownable.sol
- PairTokenManager.sol
- Proxy.sol
- ReentrancyGuard.sol
- SafeCast.sol
- SafeMath.sol
- SafeMathUInt128.sol
- Storage.sol
- TokenInit.sol
- Upgradeable.sol

- UpgradeableMaster.sol
- UpgradeGatekeeper.sol
- Utils.sol
- Verifier.sol
- VerifierExit.sol
- ZkSync.sol
- zkSyncCommitBlock.sol

The fixes were provided in commit.

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 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.

2.3 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.

3 Detailed Results

3.1 CVF-1

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

Description The protocol is a fork of a fork of zkSync, and it is not called "zkSync" anymore.
Recommendation Consider replacing "zkSync" with the actual protocol name everywhere, except for credits to the original implementation.

Listing 1:

```
/// @title zkSync main contract

contract ZkSyncCommitBlock is PairTokenManager, Storage, Config,
    Events, ReentrancyGuard {

    /// @dev This function is used to allow tokens to spend
        zkSync contract balance up to amount that is requested

    /// @notice  Withdraws tokens from zkSync contract to the
        owner

    ///          NOTE: We will call ERC20.transfer(.., _amount),
        but if according to internal logic of ERC20 token
        zkSync contract

            // and fail if token subtracted from zkSync balance
                more then `_amount` that was requested.

    /// @notice Withdraws token from ZkSync to root chain in
        case of exodus mode. User must provide proof that he
        owns funds
```

3.2 CVF-2

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

Recommendation Consider initializing the constant with the hash expression instead of a hardcoded value: bytes32 private constant EMPTY_STRING_KECCAK = keccak256("");

Listing 2:

```
bytes32 public constant EMPTY_STRING_KECCAK = 0
    xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
    ;
```

3.3 CVF-3

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 3:

```
64  function initialize(bytes calldata) external {}
```

3.4 CVF-4

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

Recommendation The arguments of 'CommitBlockInfo' type should be declared as "calldata" rather than "memory" to avoid extra copying inside the "createBlockCommitment" function.

Listing 4:

```
69  function commitOneBlock(StoredBlockInfo memory _previousBlock,
      ↪ CommitBlockInfo memory _newBlock)

104     CommitBlockInfo[] memory _newBlocksData

333 function collectOnchainOps(CommitBlockInfo memory _newBlockData)

472     CommitBlockInfo memory _newBlockData,
```

3.5 CVF-5

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

Recommendation This check is redundant if we assume that all blocks suggested by validators are intended to be consecutive.

Listing 5:

```
74  require(_newBlock.blockNumber == _previousBlock.blockNumber + 1,
      ↪ "f"); // only commit next block
```

3.6 CVF-6

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

Description Checking two conditions at once makes it harder to investigate problems.
Recommendation Consider splitting this "require" statement into two statements.

Listing 6:

```
81  require(timestampNotTooSmall && timestampNotTooBig, "h"); // New block timestamp is not valid
```

3.7 CVF-7

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

Description Using "uint32" for loop counter is less efficient that "uint256".
Recommendation Consider using "uint256" instead.

Listing 7:

```
109 for (uint32 i = 0; i < _newBlocksData.length; ++i) {

158 for (uint32 i = 0; i < blocksToRevert; ++i) {

301 for (uint32 i = 0; i < _blockExecuteData.pendingOnchainOpsPubdata.length; ++i) {
```

3.8 CVF-8

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

Description This reads and writes the "totalCommittedPriorityRequests" variable on every loop iteration.
Recommendation Consider counting committed priority requests in a local variable and update the storage only once after the loop.

Listing 8:

```
112 totalCommittedPriorityRequests += _lastCommittedBlockData.priorityOperations;
```

3.9 CVF-9

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

Description This event just logs consecutive numbers.
Recommendation Consider logging something more meaningful.

Listing 9:

```
115 emit BlockCommit(_lastCommittedBlockData.blockNumber);
```

3.10 CVF-10

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

Description Overflow is possible here.
Recommendation Consider using safe operations.

Listing 10:

```
118 totalBlocksCommitted += uint32(_newBlocksData.length);
```

3.11 CVF-11

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

Recommendation These two lines could be merged into one: require(hashStoredBlockInfo(_committedBlocs[i]) == storedBlockHashes[++currentTotalBlocksProven], "o1");

Listing 11:

```
130 require(hashStoredBlockInfo(_committedBlocks[i]) ==
      ↪ storedBlockHashes[currentTotalBlocksProven + 1], "o1");
++currentTotalBlocksProven;
```

3.12 CVF-12

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** zkSyncCommitBlock.sol

Description The code would be simpler and less error-prone is all block commitments would be guaranteed to have zeros on the highest three bits. In such case it wouldn't be necessary to mask off these bits.
Recommendation Consider changing the "createBlockCommitment" function to zero the highest three bits of all the commitments.

Listing 12:

```
133 require(_proof.commitments[i] & INPUT_MASK == uint256(
    ↪ _committedBlocks[i].commitment) & INPUT_MASK, "o"); //
    ↪ incorrect block commitment in proof
```

3.13 CVF-13

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

Description This requirement implies that sending commitments in both function arguments is redundant.

Listing 13:

```
133 require(_proof.commitments[i] & INPUT_MASK == uint256(
    ↪ _committedBlocks[i].commitment) & INPUT_MASK, "o"); //
    ↪ incorrect block commitment in proof
```

3.14 CVF-14

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

Description This variable is redundant as it is used only once.
Recommendation Consider using expression intead.

Listing 14:

```
136 bool success =
```

3.15 CVF-15

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

Recommendation These two lines could be merged into one: delete storedBlockHashes[blocksCommitted–];

Listing 15:

```
162 delete storedBlockHashes[blocksCommitted];

164 --blocksCommitted;
```

3.16 CVF-16

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

Description The first parameter was not changed in this function so it is odd to log it.

Listing 16:

```
174 emit BlocksRevert(totalBlocksExecuted, blocksCommitted);
```

3.17 CVF-17

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** zkSyncCommitBlock.sol

Description Here the "SafeMath.sub" function is used to enforce a business-level constraint. This makes the code harder to read and more error-prone.
Recommendation Consider checking business-level constraints explicitly and using "SafeMath" as a second layer of defense.

Listing 17:

```
188 pendingBalances[packedBalanceKey].balanceToWithdraw = balance.
    ↪ sub(_amount);
```

3.18 CVF-18

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

Description This function is not documented.

Listing 18:

```
function increaseBalanceToWithdraw(bytes22 _packedBalanceKey,
    ↪ uint128 _amount) internal {
```

3.19 CVF-19

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

Description This function is identical to a function declared in the "ZkSync" smart contract.
Recommendation Consider extracting to a library or a common base contract.

Listing 19:

```
function _transferERC20(
```

3.20 CVF-20

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

Description The name of the returned value is confusing and this function doesn't "withdraw" anything, but rather "transfers" (according to the function name) or "sends" according to the documentation comment.
Recommendation Consider renaming to "transferredAmount" or "sentAmount".

Listing 20:

```
) external returns (uint128 withdrawnAmount) {
```

3.21 CVF-21

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

Recommendation The type of this argument should be "IERC20".

Listing 21:

```
232 address _token,
```

3.22 CVF-22

- **Severity** Critical
- **Category** Flaw
- **Status** Fixed
- **Source** zkSyncCommitBlock.sol

Recommendation This will revert for a pair token ID.The pair tokens should be handled differently, as they ought to be minted rather than sent.

Listing 22:

```
240 uint16 tokenId = governance.validateTokenAddress(_token);
```

3.23 CVF-23

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

Description Creating a slice allocates a new bytes array and copies the data into it. This is suboptimal.
Recommendation Consider accessing the data in-place instead.

Listing 23:

```
362 bytes memory opPubData = Bytes.slice(pubData, pubdataOffset,
      ↪ DEPOSIT_BYTES);

367 bytes memory opPubData = Bytes.slice(pubData, pubdataOffset,
      ↪ CREATE_PAIR_BYTES);

372 bytes memory opPubData = Bytes.slice(pubData, pubdataOffset,
      ↪ CHANGE_PUBKEY_BYTES);

382     opPubData = Bytes.slice(pubData, pubdataOffset,
          ↪ WITHDRAW_BYTES);

384     opPubData = Bytes.slice(pubData, pubdataOffset,
          ↪ FULL_EXIT_BYTES);
```

3.24 CVF-24

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

Description This variable is redundant as it is used only once.
Recommendation Consider using an expression instead.

Listing 24:

```
bool valid = verifyChangePubkey(onchainOpData.ethWitness, op);
```

3.25 CVF-25

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

Description There is no length check for "_ethWitness".
Recommendation Consider adding such check.

Listing 25:

```
(, bytes memory signature) = Bytes.read(_ethWitness, 1, 65); //
  ↪ offset is 1 because we skip type of ChangePubkey
```

3.26 CVF-26

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

Recommendation The comparison to zero address would not be necessary if the "Utils.recoverAddressFromEthSignature" function would properly check values returned by the "ecrecover" function to be non-zero.

Listing 26:

```
return recoveredAddress == _changePk.owner && recoveredAddress
  ↪ != address(0);
```

3.27 CVF-27

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

Description The trick is not necessary here, as the public data is anyway copied when being concatenated with the offset commitment.
Recommendation Just add "hash" to this concatenation: bytes memory pubdata = abi.encodePacked(hash, _newBlockData.publicData, _offsetCommitment);

Listing 27:

```
480 bytes memory pubdata = abi.encodePacked(_newBlockData.publicData
    ↪ , _offsetCommitment);

482 /// The code below is equivalent to `commitment = sha256(abi.
    ↪ encodePacked(hash, _publicData))`

484 /// We use inline assembly instead of this concise and readable
    ↪ code in order to avoid copying of `_publicData` (which
    ↪ saves ~90 gas per transfer operation).
```

3.28 CVF-28

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

Description Solidity reserves the first 64 bytes of memory a scratch area for hash operations.
Recommendation Consider using it instead of the first work on unallocated memory. See documentation for details: https://docs.soliditylang.org/en/v0.7.0/internals/layout_in_memory.html

Listing 28:

```
491 let hashResult := mload(0x40)
```

3.29 CVF-29

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

Description This variable is used only once.
Recommendation Consider using the expression instead.

Listing 29:

```
Operations.OpType priorReqType = priorityRequests[_priorityRequestId].opType;
require(priorReqType == Operations.OpType.Deposit, "H"); // incorrect priority op type
```

```
Operations.OpType priorReqType = priorityRequests[_priorityRequestId].opType;
require(priorReqType == Operations.OpType.FullExit, "J"); // incorrect priority op type
```

3.30 CVF-30

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

Recommendation This requirement should be enforced before the loop.

Listing 30:

```
require(totalBlocksExecuted <= totalBlocksProven, "n"); // Can't execute blocks more then committed and proven currently.
```

3.31 CVF-31

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

Description This variable is redundant.
Recommendation Just give a name to the return values and use it instead.

Listing 31:

```
(bool callSuccess, ) = _to.call{gas: WITHDRAWAL_GAS_LIMIT, value: _amount}("");
```

3.32 CVF-32

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

Recommendation This calculation should be done after the "require" statements.

Listing 32:

```
bytes22 packedBalanceKey = packAddressAndTokenId(_owner,
    ↪ _tokenId);
```

3.33 CVF-33

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

Description This variable is redundant as it is used only once.
Recommendation Just use expression instead.

Listing 33:

```
bool proofCorrect =
```

3.34 CVF-34

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

Description This just increases the withdrawable balance, but doesn't transfer assets, so the user will have to call "withdrawPendingBalance" separately.
Recommendation Consider implementing an ability to perform exodus and withdraw in once transaction.

Listing 34:

```
increaseBalanceToWithdraw(packedBalanceKey, _amount);
```

3.35 CVF-35

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

Description The function name is confusing, as it increases the withdrawable balance by the given amount rather than sets it to the given value.
Recommendation Consider renaming.

Listing 35:

```
function updateBalance(address _owner, uint16 _tokenId, uint128
    ↪ _out) internal {
```

3.36 CVF-36

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

Recommendation The type of the "pair" argument should be "IUniswapV2Pair".

Listing 36:

```
function checkLpL1Balance(address pair, uint128 _lpL1Amount)
    ↪ internal {
```

3.37 CVF-37

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

Description Unlike names of other function arguments, the name of the "pair" argument is not prefixed with underscore ("_").
Recommendation Consider adding underscore for consistency.

Listing 37:

```
function checkLpL1Balance(address pair, uint128 _lpL1Amount)
    ↪ internal {
```

3.38 CVF-38

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

Description The function name is very confusing. It looks like a checker function with no side effects, while actually it burns tokens.
Recommendation Consder renaming.

Listing 38:

```
function checkLpL1Balance(address pair, uint128 _lpL1Amount)
    ↪ internal {
```

3.39 CVF-39

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

Recommendation The cast is redundant as '_lpL1Amount' is already 128-bit.

Listing 39:

```
pairmanager.burn(address(pair), msg.sender, SafeCast.toUint128(
    ↪ _lpL1Amount)); //
```

3.40 CVF-40

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

Recommendation The type of the "_pairAccount" argument should be "IUniswapV2Pair".

Listing 40:

```
function checkPairAccount(address _pairAccount, uint16[] memory
    ↪ _tokenIds) internal view {
```

3.41 CVF-41

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

Description There is no length check for the "_tokenIds" argument, while it seems that this array should always have three elements.
Recommendation Consider ether adding a length check or just use a fixed-sized array of 3 elements.

Listing 41:

```
619 function checkPairAccount(address _pairAccount, uint16[] memory
    ↪ _tokenIds) internal view {

642 function lpExit(StoredBlockInfo memory _storedBlockInfo, uint32
    ↪ [] calldata _accountIds, address[] calldata _addresses,
    ↪ uint16[] calldata _tokenIds, uint128[] calldata _amounts,
    ↪ uint256[] calldata _proof) external nonReentrant {
```

3.42 CVF-42

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

Description This assignment should be done only if "_tokenIds[0]" is not zero.

Listing 42:

```
625 address _token0 = governance.tokenAddresses(_tokenIds[0]);
```

3.43 CVF-43

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

Description This assignment should be done only if "_tokenIds[1]" is not zero.

Listing 43:

```
631 address _token1 = governance.tokenAddresses(_tokenIds[1]);
```

3.44 CVF-44

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

Description There are no length checks for the "_accountIds", "_addresses", "_tokenIds", and "_amounts" arrays, while all these arrays are supposed to have some predefined lengths.
Recommendation Consider ether adding proper length checks or even better to use static-sized arrays instead.

Listing 44:

```
function lpExit(StoredBlockInfo memory _storedBlockInfo, uint32
  ↪ [] calldata _accountIds, address[] calldata _addresses,
  ↪ uint16[] calldata _tokenIds, uint128[] calldata _amounts,
  ↪ uint256[] calldata _proof) external nonReentrant {
```

3.45 CVF-45

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

Recommendation This should be in a documentation comment before the function.

Listing 45:

```
/* data format:
    StoredBlockInfo _storedBlockInfo
     _owner_id = _accountIds[0]
     _pair_acc_id = _accountIds[1]
     _owner_addr = _addresses[0]
     _pair_acc_addr = _addresses[1]
     _token0_id = _tokenIds[0]
     _token1_id = _tokenIds[1]
     _lp_token_id = _tokenIds[2]
     _token0_amount = _amounts[0]
     _token1_amount = _amounts[1]
     _lp_L1_amount = _amounts[2]

*/
```

3.46 CVF-46

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

Description This comment is not relevant to the next line but seems to belong to the line after it.
Recommendation Consider moving the comment one line down.

Listing 46:

```
//check root hash
```

3.47 CVF-47

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

Description The "_addresses[0]" value is redundant as it could be derived form "msg.sender".
Recommendation Consider shrinking the "_addresses" array by one.

Listing 47:

```
require(msg.sender == _addresses[0], "le2");
```

3.48 CVF-48

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

Description This line looks like a check, while it actually burn tokens.
Recommendation Consider explaining this fact in a comment.

Listing 48:

```
checkLpL1Balance(_addresses[1], _amounts[2]);
```

3.49 CVF-49

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

Description The protocol is a fork of a fork of zkSync, and it is not called "zkSync" anymore.
Recommendation Consider replacing "zkSync" with the actual protocol name everywhere, except for credits to the original implementation.

Listing 49:

```
/// @title zkSync main contract

contract ZkSync is PairTokenManager, UpgradeableMaster, Storage,
    ↪ Config, Events, ReentrancyGuard {

    /// @notice zkSync contract initialization. Can be external
        ↪ because Proxy contract intercepts illegal calls of
        ↪ this function.

    /// @notice zkSync contract upgrade. Can be external because
        ↪  Proxy contract intercepts illegal calls of this
        ↪ function.

    /// @dev This function is used to allow tokens to spend
        ↪ zkSync contract balance up to amount that is requested

    /// @param _zkSyncAddress The receiver Layer 2 address
    function depositETH(address _zkSyncAddress) external payable
        ↪  {

        registerDeposit(0, SafeCast.toUint128(msg.value),
            ↪ _zkSyncAddress);

    /// @notice Returns amount of tokens that can be withdrawn
        ↪ by `address` from zkSync contract

    /// @notice Returns amount of tokens that can be withdrawn
        ↪ by `address` from zkSync contract
```

3.50 CVF-50

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

Recommendation Consider initializing the constant with the hash expression instead of a hardcoded value: bytes32 private constant EMPTY_STRING_KECCAK = keccak256("");.

Listing 50:

```
bytes32 private constant EMPTY_STRING_KECCAK = 0
    ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
    ↪ ;
```

3.51 CVF-51

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

Recommendation The arguments type should be "IERC20".

Listing 51:

```
33 function createPair(address _tokenA, address _tokenB) external {

59 function createETHPair(address _tokenERC20) external {
```

3.52 CVF-52

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** ZkSync.sol

Description It is not ensured that the tokens are different.
Recommendation Consider adding an explicit check for this.

Listing 52:

```
33 function createPair(address _tokenA, address _tokenB) external {
```

3.53 CVF-53

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

Description It is not ensured that neither token is zero. IT is currently possible to add a token with zero address and obtain a non-zero token ID for it. Adding a pair against such token will effectively create a pair against ether in L! and a pair against non-zero token ID in L2.
Recommendation Consider adding an explicit check that neither token is zero.

Listing 53:

```
function createPair(address _tokenA, address _tokenB) external {
```

3.54 CVF-54

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

Description This check is redundant, as UniswapV2Factory contract, known as "pairManager" doesn't use zero pair address to indicate an error, but just reverts.
Recommendation Consider removing this check.

Listing 54:

```
require(pair != address(0), "pair is invalid");

require(pair != address(0), "pair is invalid");
```

3.55 CVF-55

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

Description This call wouldn't be necessary if the "addPairToken" function would return the ID of the new pair token.

Listing 55:

```
validatePairTokenAddress(pair),

        validatePairTokenAddress(pair),
```

3.56 CVF-56

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

Description It is not ensured that the token address is not zero.
Recommendation Consider adding such check.

Listing 56:

```
function createETHPair(address _tokenERC20) external {
```

3.57 CVF-57

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

Recommendation The type of the "_pair" argument should be "IUniswapV2Pair".

Listing 57:

```
function registerCreatePair(uint16 _tokenA, uint16 _tokenB,
    ↪ uint16 _tokenPair, address _pair) internal {
```

3.58 CVF-58

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

Description Reordering token IDs is redundant. The tokens are already ordered by their contract addresses which guarantees pair uniqueness.

Listing 58:

```
(uint16 token0, uint16 token1) = _tokenA < _tokenB ? (_tokenA,
    ↪ _tokenB) : (_tokenB, _tokenA);
```

3.59 CVF-59

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 59:

```
function upgradeNoticePeriodStarted() external override {}
```

3.60 CVF-60

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

Recommendation The type of this variable should be "Governance".

Listing 60:

```
address _governanceAddress,
```

3.61 CVF-61

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

Recommendation The type of this variable should be "Verifier".

Listing 61:

```
address _verifierAddress,
```

3.62 CVF-62

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

Recommendation The type of this variable should be "VerifierExit".

Listing 62:

```
address _verifierExitAddress,
```

3.63 CVF-63

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

Recommendation The type of this variable should be "UniswapV2Factory".

Listing 63:

```
address _pairManagerAddress,
```

3.64 CVF-64

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

Recommendation The type of this variable should be "zkSyncCommitBlock".

Listing 64:

```
address _zkSyncCommitBlockAddress,
```

3.65 CVF-65

- **Severity** Critical
- **Category** Flaw
- **Status** Fixed
- **Source** ZkSync.sol

Description The contract performs delegate calls to the address set here. A malicious user may call "initialize" directly on the contract (bypassing proxy) and pass a special contract address as the "zkSyncCommitBlockAddress"argument. Then, the user may call a function on the contract that will be delegated to the special contract. If the special contract will execute the "SELFDESTRUCT" opcode, the main ZkSync contract will be destroyed breaking up the protocol completely.
Recommendation Consider forbidding direct calls to "initialize". One way to do this is to store the contract's own address into an immutable variable inside constructor, and then ensuring, that the value of "this" is different from the stored address. Alternatively, consider hardcoding the zkSyncCommitBock address, or passing it as a constructor argument and storing in an immutable variable.

Listing 65:

```
zkSyncCommitBlockAddress = _zkSyncCommitBlockAddress;
```

3.66 CVF-66

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

Description Overflow is possible here.
Recommendation Consider using safe addition and conversion.

Listing 66:

```
uint64 expirationBlock = uint64(block.number +
    ↪ PRIORITY_EXPIRATION);
```

3.67 CVF-67

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

Description This version isn't supposed to be used as an upgrade.
Recommendation Consider removing this function or just reverting in its body.

Listing 67:

```
function upgrade(bytes calldata upgradeParameters) external
    ↪ nonReentrant {
```

3.68 CVF-68

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

Description The "totalBlocksCommitted" variable is read twice.
Recommendation Consider reading once and reusing.

Listing 68:

```
require(totalBlocksCommitted == totalBlocksProven, "wq1"); //
    ↪ All the blocks must be proven
require(totalBlocksCommitted == totalBlocksExecuted, "w12"); //
    ↪ All the blocks must be executed
```

3.69 CVF-69

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

Description The "totalBlocksExecuted" variable is read twice.
Recommendation Consider reading once and reusing.

Listing 69:

```
196 require(totalBlocksCommitted == totalBlocksExecuted, "w12"); //
    ↪ All the blocks must be executed

202 storedBlockHashes[totalBlocksExecuted] = hashStoredBlockInfo(
    ↪ lastBlockInfo);
```

3.70 CVF-70

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

Description This basically hardcodes the zkSyncCommitBlock address, which is inconsistent with how the address is obtained in the "initialize" function.
Recommendation As an alternative for hardcoding, consider passing the address as a constructor argument and storing in an immutable variable.

Listing 70:

```
204 zkSyncCommitBlockAddress = address($$(
    ↪ NEW_ZKSYNC_COMMIT_BLOCK_ADDRESS));
```

3.71 CVF-71

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

Description The name of the returned value is confusing and this function doesn't "withdraw" anything, but rather "transfers" (according to the function name) or "sends" according to the documentation comment.
Recommendation Consider renaming to "transferredAmount" or "sentAmount".

Listing 71:

```
217 function _transferERC20(
```

3.72 CVF-72

- **Severity** Moderate
- **Category** Procedural
- **Status** Opened
- **Source** ZkSync.sol

Recommendation This check doesn't make sense and should be removed. The "toProcess" value counts all types of priority requests, while the "_depositPubdata" array contains data for deposits only. In case there are open priority requests other than deposits, the caller will need to add extra elements to the "_depositPubdata" just to satisfy this condition, effectively wasting gas.

Listing 72:

```
require(toProcess == _depositsPubdata.length, "A");
```

3.73 CVF-73

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

Description Operations with types narrower than 256 bits are more expensive in general, that with 256-bit words.
Recommendation Consider using uint256 instead of uint64 for the counters.

Listing 73:

```
uint64 currentDepositIdx = 0;
for (uint64 id = firstPriorityRequestId; id <
    ↪ firstPriorityRequestId + toProcess; id++) {
```

3.74 CVF-74

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

Description The expression "firstPriorityRequestId + toProcess" is calculated on every loop iteration.
Recommendation Consider calculating once and reusing.

Listing 74:

```
for (uint64 id = firstPriorityRequestId; id <
    ↪ firstPriorityRequestId + toProcess; id++) {
```

3.75 CVF-75

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

Description The expression "priorityRequests[id]" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 75:

```
if (priorityRequests[id].opType == Operations.OpType.Deposit) {

    require(Utils.hashBytesToBytes20(depositPubdata) == priorityRequests[id].hashedPubData, "a");

delete priorityRequests[id];
```

3.76 CVF-76

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

Recommendation These lines could be merged together like this: bytes memory depositPubdata = _depositPubdata[currentDepositIdx++];

Listing 76:

```
bytes memory depositPubdata = _depositsPubdata[currentDepositIdx];

++currentDepositIdx;
```

3.77 CVF-77

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

Description The name can be confused with a name of some ZkSync contracts.
Recommendation Consider renaming.

Listing 77:

```
/// @param _zkSyncAddress The receiver Layer 2 address
```

3.78 CVF-78

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

Description The word "franklin" here is confusing, as it is not used as the protocol name anymore.
Recommendation Consider renaming.

Listing 78:

```
/// @param _franklinAddr Receiver Layer 2 address
function depositERC20(IERC20 _token, uint104 _amount, address
    ↪ _franklinAddr) external nonReentrant {
```

3.79 CVF-79

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

Description The assignment is redundant as 'lpTokenId' has been already set to the returned value.
Recommendation Consider extracting a boolean check from ‘validatePairTokenAddress‘.

Listing 79:

```
// Get token id by its address
```

3.80 CVF-80

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

Description These logic is coded twice.
Recommendation Consider refactoring the code to remove code duplication.

Listing 80:

```
balance_before = _token.balanceOf(msg.sender);

balance_after = _token.balanceOf(msg.sender);
deposit_amount = SafeCast.toUint128(balance_before.sub(
    ↪ balance_after));

balance_before = _token.balanceOf(address(this));

balance_after = _token.balanceOf(address(this));
deposit_amount = SafeCast.toUint128(balance_after.sub(
    ↪ balance_before));
```

3.81 CVF-81

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

Recommendation The type of the "_token" argument should be "IERC20".

Listing 81:

```
function getPendingBalance(address _address, address _token)
    ↪ public view returns (uint128) {

function requestFullExit(uint32 _accountId, address _token,
    ↪ uint32 _pairAccountId) public nonReentrant {
```

3.82 CVF-82

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

Description The comment seems incomplete.

Listing 82:

```
/// @param _pairAccountId pair account id, only use when _token
    ↪ is pair address and a
```

3.83 CVF-83

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

Description This assignment should be done only if "_token" is not zero.

Listing 83:

```
341 uint16 lpTokenId = tokenIds[_token];
```

3.84 CVF-84

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** ZkSync.sol

Description Probably it should be 'else if' here. Otherwise a request to full exit with ether will have lpTokenId equal 0 and revert at line 348

Listing 84:

```
345 } if (lpTokenId == 0) {
```

3.85 CVF-85

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

Description This assignment is needed only in case the pending balances slot is currently zero.

Listing 85:

```
372 pendingBalances[packedBalanceKey].gasReserveValue =
      ↪ FILLED_GAS_RESERVE_VALUE;
```

3.86 CVF-86

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

Description The expression "priorityRequests[firstPriorityRequestId].expirationBlock" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 86:

```
381 block.number >= priorityRequests[firstPriorityRequestId].
     ↪ expirationBlock &&
       priorityRequests[firstPriorityRequestId].expirationBlock !=
         ↪ 0;
```

3.87 CVF-87

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

Description This expression potentially accesses a non-existing priority requests.
Recommendation Consider rewriting as: totalOpenPriorityRequests > 0 && block.number >= priorityRequests[firstPriorityRequestId].expirationBlock;

Listing 87:

```
381 block.number >= priorityRequests[firstPriorityRequestId].
     ↪ expirationBlock &&
       priorityRequests[firstPriorityRequestId].expirationBlock !=
         ↪ 0;
```

3.88 CVF-88

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

Recommendation This check should be done at the very beginning of the function.

Listing 88:

```
384 if (!exodusMode) {
```

3.89 CVF-89

- **Severity** Critical
- **Category** Flaw
- **Status** Fixed
- **Source** ZkSync.sol

Description A malicious user may call the "initialize" function directly, i.e. bypassing proxy, to set the "zkSyncCommitBlockAddress" value to the address of a special smart contract that self-destructs when being called. Then the malicious user may trigger the fallback function directly, effectively destroying the main ZkSync contract. This will most probably destroy the whole protocol and make user funds lost forever.
Recommendation Consider forbidding calling the "initialize" and fallback functions directly. This could be achieved by storing the contract's own address into an immutable variable inside the constructor, and then ensuring inside the "initialize" and fallback functions that the "this" value differs from the stored own address.

Listing 89:

```
address nextAddress = zkSyncCommitBlockAddress;

    let result := delegatecall(gas(), nextAddress, 0, calldatasize(), 0, 0)
```

3.90 CVF-90

- **Severity** Major
- **Category** Suboptimal
- **Status** Info
- **Source** ZkSync.sol

Description Chaining smart contracts like this is suboptimal, error-prone, and thus discouraged. The proper way is to define all the entry points in one contract and then delegate some work to libraries deployed separately.
Client Comment Not an issue after fix CVF-65

Listing 90:

```
let result := delegatecall(gas(), nextAddress, 0, calldatasize(), 0, 0)
```

3.91 CVF-91

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

Recommendation Just initialize the constant using the expression given in the comment instead of a hardcoded value. Solidity compiler is smart enough to calculate the expression at compile time.

Listing 91:

```
/// @dev Storage position of "target" (actual implementation address: keccak256('eip1967.proxy.implementation') - 1)
bytes32 private constant targetPosition = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;
```

3.92 CVF-92

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

Description Reading the target address from the storage is suboptimal, as this address is available in a constructor argument.
Recommendation Consider using it from there.

Listing 92:

```
getTarget().delegatecall(abi.encodeWithSignature("initialize(bytes)", targetInitializationParameters));

getTarget().delegatecall(abi.encodeWithSignature("upgrade(bytes)", newTargetUpgradeParameters));
```

3.93 CVF-93

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

Recommendation These variables are redundant. Just use the “targetPosition” constant directly in the assembly blocks.

Listing 93:

```
bytes32 position = targetPosition;

bytes32 position = targetPosition;
```

3.94 CVF-94

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

Description This function should log an event.

Listing 94:

```
function upgradeTarget(address newTarget, bytes calldata
    ↪ newTargetUpgradeParameters) external override {
```

3.95 CVF-95

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

Recommendation Most of this assembly block could be rewritten in plain Solidity.

Listing 95:

```
assembly {
```

3.96 CVF-96

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

Description This line is redundant, as the contract will not use memory after the call.
Recommendation Just copy the call data starting at zero memory address.

Listing 96:

```
let ptr := mload(0x40)
```

3.97 CVF-97

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

Description This function is redundant as a payable fallback function already exists with the same body.

Listing 97:

```
receive() external payable {
```

3.98 CVF-98

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

Description This function could just call the "_fallback" function, as they basically just forward the call as is to the target and then forward the result back.

Listing 98:

```
function getNoticePeriod() external override returns (uint256) {

function upgradeNoticePeriodStarted() external override {

function upgradePreparationStarted() external override {

function upgradeCanceled() external override {

function upgradeFinishes() external override {

function isReadyForUpgrade() external override returns (bool) {
```

3.99 CVF-99

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

Description These checks drop the original error message got from the target and replaces it with mostly useless message.
Recommendation Consider propagating the original message instead of replacing it.

Listing 99:

```
require(success, "unp11"); // unp11 - upgradeNoticePeriod
    ↪ delegatecall failed

require(success, "nps11"); // nps11 - upgradeNoticePeriodStarted
    ↪  delegatecall failed

require(success, "ups11"); // ups11 - upgradePreparationStarted
    ↪ delegatecall failed

require(success, "puc11"); // puc11 - upgradeCanceled
    ↪ delegatecall failed

require(success, "puf11"); // puf11 - upgradeFinishes
    ↪ delegatecall failed

require(success, "rfu11"); // rfu11 - readyForUpgrade
    ↪ delegatecall failed
```

3.100 CVF-100

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

Description There is no access modifier specified for these constants, so internal access will be used by default.
Recommendation Consider explicitly specifying access level.

Listing 100:

```
uint16 constant MAX_AMOUNT_OF_PAIR_TOKENS = 1920;
uint16 constant PAIR_TOKEN_START_ID = 128;
```

3.101 CVF-101

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

Recommendation This constant should be related to MAX_AMOUNT_OF_REGISTERED_TOKENS.

Listing 101:

```
uint16 constant PAIR_TOKEN_START_ID = 128;
```

3.102 CVF-102

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

Description Using types narrower than 256 bit doesn't save gas and could actually be more expensive.
Recommendation Consider using uint256 instead of uint16.

Listing 102:

```
uint16 public totalPairTokens;
```

3.103 CVF-103

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

Recommendation The value type for this mapping should be "IUniswapV2Pair".

Listing 103:

```
mapping(uint16 => address) public tokenAddresses;
```

3.104 CVF-104

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

Recommendation The key type for this mapping should be "IUniswapV2Pair".

Listing 104:

```
mapping(address => uint16) public tokenIds;
```

3.105 CVF-105

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

Recommendation The type of this parameter should be “IUniswapV2Pair”.

Listing 105:

```
address indexed token,
```

3.106 CVF-106

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

Recommendation The types of arguments should be “IUniswapV2Pair”.

Listing 106:

```
function addPairToken(address _token) internal {

function validatePairTokenAddress(address _tokenAddr) public
    ↪ view returns (uint16) {
```

3.107 CVF-107

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

Description This function should return the ID of the new pair token.

Listing 107:

```
function addPairToken(address _token) internal {
```

3.108 CVF-108

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

Description The "totalPairToken" variable is read from the storage twice.
Recommendation Consider reading once and reusing.

Listing 108:

```
require(totalPairTokens < MAX_AMOUNT_OF_PAIR_TOKENS, "pan2"); //
    ↪ no free identifiers for tokens

totalPairTokens++;
```

3.109 CVF-109

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

Recommendation It would be more efficient to store the next pair token ID instead of the pair tokens count.

Listing 109:

```
uint16 newPairTokenId = PAIR_TOKEN_START_ID + totalPairTokens;
```

3.110 CVF-110

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** PairTokenManager.sol

Description This function does not check that tokenID is not smaller than PAIR_TOKEN_START_ID.

Listing 110:

```
function validatePairTokenAddress(address _tokenAddr) public
    ↪ view returns (uint16) {
```

3.111 CVF-111

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

Description This check is redundant, as pair token IDs written into the “tokenIds” mapping never violate it.

Listing 111:

```
require(tokenId < (PAIR_TOKEN_START_ID +
    ↪ MAX_AMOUNT_OF_PAIR_TOKENS), "pms4");
```

3.112 CVF-112

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

Recommendation The type of the “token” parameter should be “IERC20”.

Listing 112:

```
event NewToken(address indexed token, uint16 indexed tokenId);
```

3.113 CVF-113

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

Recommendation It would be cheaper to have two separate events for validator activation and deactivation.

Listing 113:

```
event ValidatorStatusUpdate(address indexed validatorAddress,
    ↪ bool isActive);
```

3.114 CVF-114

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

Recommendation It would be cheaper to have two separate events for token pause and unpause.

Listing 114:

```
event TokenPausedUpdate(address indexed token, bool paused);
```

3.115 CVF-115

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

Recommendation The value type for this mapping should be "IERC20".

Listing 115:

```
mapping(uint16 => address) public tokenAddresses;
```

3.116 CVF-116

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

Recommendation The key type for this mapping should be "IERC20".

Listing 116:

```
mapping(address => uint16) public tokenIds;
```

3.117 CVF-117

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

Recommendation A bitmap of 256 words would be more efficient: uint256 [256] public pausedTokensBitmap;

Listing 117:

```
mapping(uint16 => bool) public pausedTokens;
```

3.118 CVF-118

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

Description This function should emit the "NewGovernor" event to make it possible to restore the full governor change history from the events log.

Listing 118:

```
function initialize(bytes calldata initializationParameters)
    ↪ external {
```

3.119 CVF-119

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

Description This variable is redundant as its value is used only once.
Recommendation Consider assigning directly to the “networkGovernor” storager variable.

Listing 119:

```
address _networkGovernor = abi.decode(initializationParameters,
    ↪ (address));
```

3.120 CVF-120

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

Description Once this version is not supposed to be used as an upgrade for any prior version, this function is redundant.
Recommendation Consider removing it or putting “revert()” into its body.

Listing 120:

```
function upgrade(bytes calldata upgradeParameters) external {}
```

3.121 CVF-121

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

Recommendation The type of the “_token” argument should be “IERC20”.

Listing 121:

```
function addToken(address _token) external {
```

3.122 CVF-122

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

Description There is no check that "_token" is not zero, however zero token address has a special meaning as it is used when creating pairs against ether. So, the governance may add zero token address and obtain a non-zero token ID for this address. Then, the governance may create a pair with some real (non-zero) token address against zero token address. Such token will be created against ether in L1, but will be created against non-zero token ID in L2.
Recommendation Consider explicitly forbidding adding a token with zero address.

Listing 122:

```
66 function addToken(address _token) external {
```

3.123 CVF-123

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

Description The "totalTokens" storage variable is read three times here.
Recommendation Consider reading once like this: uint16 newTokenId; require ((newTokenId = ++_totalTokens) <= MAX_AMOUNT_OF_REGISTERED_TOKENS);

Listing 123:

```
69 require(totalTokens < MAX_AMOUNT_OF_REGISTERED_TOKENS, "1f"); //
    ↪  no free identifiers for tokens

71 totalTokens++;
   uint16 newTokenId = totalTokens; // it is not `totalTokens - 1`
    ↪ because tokenId = 0 is reserved for eth
```

3.124 CVF-124

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

Recommendation The type of the "_tokenAddr" argument should be "IERC20".

Listing 124:

```
84 function setTokenPaused(address _tokenAddr, bool _tokenPaused)
    ↪ external {
```

3.125 CVF-125

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

Description The "this." syntax performs an external call which is more expensive than internal one.
Recommendation Consider using internal call here.

Listing 125:

```
uint16 tokenId = this.validateTokenAddress(_tokenAddr);
```

3.126 CVF-126

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

Description The expression "pausedTokens[tokenId]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 126:

```
if (pausedTokens[tokenId] != _tokenPaused) {
    pausedTokens[tokenId] = _tokenPaused;
```

3.127 CVF-127

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

Description The expression "validators[_validator]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 127:

```
if (validators[_validator] != _active) {
    validators[_validator] = _active;
```

3.128 CVF-128

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

Description This function treats zero token ID as a valid one.
Recommendation Probably not an issue, but consider documenting this behavior.

Listing 128:

```
function isValidTokenId(uint16 _tokenId) external view returns (
    ↪ bool) {
```

3.129 CVF-129

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** UniswapV2Pair.sol

Description The name is confusing, as this contract isn't actually a Uniswap V2 pair contract, but just partially emulates the Uniswap V2 pair API.
Recommendation Consider renaming this contract as well as other related contracts, replacing the word "Uniswap" with "L2Swap". A credit to Uniswap could be left in comments.

Listing 129:

```
contract UniswapV2Pair is UniswapV2ERC20 {
```

3.130 CVF-130

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** UniswapV2Pair.sol

Recommendation This variable should have type "UniswapV2Factory" or "IUniswapV2Factory".

Listing 130:

```
address public factory;
```

3.131 CVF-131

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** UniswapV2Pair.sol

Recommendation These variables should have type "IERC20".

Listing 131:

```
address public token0;
address public token1;
```

3.132 CVF-132

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** UniswapV2Pair.sol

Recommendation This variable probably should be declared immutable.

Listing 132:

```
address public factory;
```

3.133 CVF-133

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** UniswapV2Pair.sol

Description There is no protection from calling this function twice, so the function trusts the factory to not call it again.
Recommendation Consider adding such protection explicitly to make the code more defensive.

Listing 133:

```
// called once by the factory at time of deployment
```

3.134 CVF-134

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** UniswapV2Pair.sol

Recommendation The type of arguments should be "IERC20".

Listing 134:

```
function initialize(address _token0, address _token1) external {
```

3.135 CVF-135

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** UniswapV2Pair.sol

Recommendation The first argument should ne named “from” for consistency with the “UniswapV2ERC20.sol” smart contract.

Listing 135:

```
function burn(address to, uint256 amount) external lock {
```

3.136 CVF-136

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

Description The correctness of the value of this argument is not guaranteed by the contract. Probably not an issue.

Listing 136:

```
// `_feeAccountAddress` argument is not used by the constructor
    ↪ itself, but it's important to have this
// information as a part of a transaction, since this
    ↪ transaction can be used for restoring the tree
// state. By including this address to the list of arguments, we
    ↪ 're making ourselves able to restore
// genesis state, as the very first account in tree is a fee
    ↪ account, and we need its address before
// we're able to start recovering the data from the Ethereum
    ↪ blockchain.
```

3.137 CVF-137

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

Description The parameter name is irrelevant to the semantics.
Recommendation Consider calling it ‘salt‘.

Listing 137:

```
// `_feeAccountAddress` argument is not used by the constructor
    ↪ itself, but it's important to have this
// information as a part of a transaction, since this
    ↪ transaction can be used for restoring the tree
// state. By including this address to the list of arguments, we
    ↪ 're making ourselves able to restore
// genesis state, as the very first account in tree is a fee
    ↪ account, and we need its address before
// we're able to start recovering the data from the Ethereum
    ↪ blockchain.
```

3.138 CVF-138

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

Recommendation The name is confusing as in other functions it is just ‘validator‘

Listing 138:

```
address _firstValidator,
```

3.139 CVF-139

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

Description This checks are redundant as it can be still possible to supply dead address.

Listing 139:

```
require(_firstValidator != address(0));
require(_governor != address(0));
require(_feeAccountAddress != address(0));
```

3.140 CVF-140

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

Recommendation The types of parameters of this event should be: "Governance", "ZkSync", "Verifier", "VerifierExit", "UniswapV2Factory", and "UpgradeGatekeeper".

Listing 140:

```
53 event Addresses(address governance, address zksync, address
   ↪ verifier, address verifier_exit, address pair, address
   ↪ gatekeeper);
```

3.141 CVF-141

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

Description This function is quite simple and is used only once.
Recommendation Consider removing it and putting its logit directly where it is called from. This would make the deployment logic easier to read.

Listing 141:

```
94 function finalizeGovernance(
```

3.142 CVF-142

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

Description The function "getTokens" returns an empty array. It is supposed to be replaces with another code just before the production deployment.
Recommendation If so, consider passing the initial list of tokens are a constructor argument, to make it unnecessary to modify the code.

Listing 142:

```
99 address[] memory tokens = getTokens();
```

3.143 CVF-143

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 143:

```
function initialize(bytes calldata) external {}
```

3.144 CVF-144

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

Description This version is not supposed to be used as an upgrade.
Recommendation Consider removing this function or putting "revert();" into its body.

Listing 144:

```
function upgrade(bytes calldata upgradeParameters) external {}
```

3.145 CVF-145

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

Description The conversion to "uint256" effectively does nothing here, as the converted value is already 256 bits wide.
Recommendation Consider removing the conversion.

Listing 145:

```
sha256(abi.encodePacked(uint256(_rootHash) , _accountId, _owner,
    ↪  _tokenId, _amount));
```

3.146 CVF-146

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

Recommendation As in ‘Verifier.sol‘, clearing extra bits makes the proof malleable.

Listing 146:

```
inputs[0] = uint256(commitment) & INPUT_MASK;
```

3.147 CVF-147

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

Recommendation This function is very inefficient. It could be simplified and optimized as: return abi.encodePacked (param1, param2);

Listing 147:

```
function concatBytes(bytes memory param1, bytes memory param2)
    ↪ public pure returns (bytes memory) {
```

3.148 CVF-148

- **Severity** Major
- **Category** Flaw
- **Status** Info
- **Source** VerifierExit.sol

Description The ‘concatBytes‘ is not collision resistant, so that if some bytes from the end of ‘_account_data‘ are put into the prefix of ‘_token_data‘, the check will pass.
Recommendation Check explicitly that both arrays have certain length.
Client Comment The caller 'lpExit' will given bytes with correct certain length.

Listing 148:

```
bytes memory commit_data = concatBytes(_account_data,
    ↪ _token_data);
```

3.149 CVF-149

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

Description The condition here don't depend on user input or contract state, so it is basically constant.
Recommendation Consider removing this check or turning it into an assert.

Listing 149:

```
require(vk.num_inputs == inputs.length);
```

3.150 CVF-150

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 150:

```
function initialize(bytes calldata) external {}
```

3.151 CVF-151

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

Description This version is not supposed to be used as an upgrade.
Recommendation Consider removing this function or putting "revert();" into its body.

Listing 151:

```
function upgrade(bytes calldata upgradeParameters) external {}
```

3.152 CVF-152

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

Description It would be less error-prone to check that the inputs don't have any bits set outside the mask, instead of clearing such bits. The clearing just makes the proof malleable.
Recommendation Consider replacing these lines with the following: require (_individual_vks_inputs[i] & ĨNPUT_MASK == 0);

Listing 152:

```
uint256 commitment = _individual_vks_inputs[i];
_individual_vks_inputs[i] = commitment & INPUT_MASK;
```

3.153 CVF-153

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** Verifier.sol

Recommendation These lines could be simplified like this: _individual_vks_inputs [i] &= INPUT_MASK;

Listing 153:

```
uint256 commitment = _individual_vks_inputs[i];
_individual_vks_inputs[i] = commitment & INPUT_MASK;
```

3.154 CVF-154

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

Description Overflow is possible here.
Recommendation Consider using safe conversion.

Listing 154:

```
VerificationKey memory vk = getVkAggregated(uint32(_vkIndexes.
    ↪ length));
```

3.155 CVF-155

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

Recommendation This variable should be declared as "immutable".

Listing 155:

```
UpgradeableMaster public mainContract;
```

3.156 CVF-156

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

Recommendation The argument type should be "Upgradeable".

Listing 156:

```
function addUpgradeable(address addr) external {
```

3.157 CVF-157

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

Description The current main contract is called during the upgrade procedure. This could make it impossible to fix a broken version of the main contract via an upgrade.
Recommendation Consider limiting gas for the main contract invocations, catching exception from such invocations and allowing the governance to force upgrade even in case the main contract doesn't behave properly.

Listing 157:

```
uint256 noticePeriod = mainContract.getNoticePeriod();
mainContract.upgradeNoticePeriodStarted();

    mainContract.upgradePreparationStarted();

require(mainContract.isReadyForUpgrade(), "fpu13"); // fpu13 -
    ↪ main contract is not ready for upgrade
mainContract.upgradeFinishes();
```

3.158 CVF-158

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

Recommendation In case there are many targets to upgrade, it would be cheaper to store only the hash of the "newTargets" arrays and to pass the actual "newTarget" values again into the "finishUpgrade" function.

Listing 158:

```
68 nextTargets = newTargets;
```

3.159 CVF-159

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

Recommendation It would be more logical to use "versionId + 1" here instead of "versionId", as "versionId" is the current version, and "versionId + 1" is the version to be deployed during the upgrade.

Listing 159:

```
69 emit NoticePeriodStart(versionId, newTargets, noticePeriod);

81 emit UpgradeCancel(versionId);

93     emit PreparationStart(versionId);
```

3.160 CVF-160

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

Description A broken main contract may throw from this call effectively forbidding cancelling an upgrade (and thus scheduling another upgrade).
Recommendation Consider limiting gas for this invocation,catching exceptions from it and allowing the governance to cancel an upgrade even in case the main contract doesn't behave properly.

Listing 160:

```
77 mainContract.upgradeCanceled();
```

3.161 CVF-161

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

Description Here the upgrade parameters can be chosen at the very end of the notice period giving the users little to no time to react.
Recommendation Consider allowing upgrades with parameters fixed before the notice period starts.

Listing 161:

```
function finishUpgrade(bytes[] calldata targetsUpgradeParameters
    ↪ ) external {
```

3.162 CVF-162

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

Description This enum doesn't contain a value for the "forced exit" operation.

Listing 162:

```
enum OpType {
```

3.163 CVF-163

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

Description Using types narrower than 256 bits for constants doesn't increase performance, and actually may even decrease it. Also, it makes the code more error-prone, as arithmetic operations with such constants could be performed in narrowed type terms, thus making overflow more probable.
Recommendation Consider using uint256 type for all the constants with quantity semantics.

Listing 163:

```
uint8 constant OP_TYPE_BYTES = 1;
uint8 constant TOKEN_BYTES = 2;
uint8 constant PUBKEY_BYTES = 32;
uint8 constant NONCE_BYTES = 4;
uint8 constant PUBKEY_HASH_BYTES = 20;
uint8 constant ADDRESS_BYTES = 20;

uint8 constant FEE_BYTES = 2;

uint8 constant ACCOUNT_ID_BYTES = 4;
uint8 constant AMOUNT_BYTES = 16;

uint8 constant SIGNATURE_BYTES = 64;
```

3.164 CVF-164

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

Description There is no access level specified for these constants, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 164:

```
32 uint8 constant OP_TYPE_BYTES = 1;
   uint8 constant TOKEN_BYTES = 2;
   uint8 constant PUBKEY_BYTES = 32;
   uint8 constant NONCE_BYTES = 4;
   uint8 constant PUBKEY_HASH_BYTES = 20;
   uint8 constant ADDRESS_BYTES = 20;

39 uint8 constant FEE_BYTES = 2;

41 uint8 constant ACCOUNT_ID_BYTES = 4;
   uint8 constant AMOUNT_BYTES = 16;

44 uint8 constant SIGNATURE_BYTES = 64;
```

3.165 CVF-165

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

Recommendation Probably should be TOKEN_ID_BYTES.

Listing 165:

```
33 uint8 constant TOKEN_BYTES = 2;
```

3.166 CVF-166

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

Description There is no separate constant for the length of pair account ID.
Recommendation Consider adding one or a comment why it equals the length of the regular account ID.

Listing 166:

```
41 uint8 constant ACCOUNT_ID_BYTES = 4;
```

3.167 CVF-167

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

Description These "NEW ADD" comments don't help reading the code. Git history is a better place to look for change history.
Recommendation Consider removing these comments.

Listing 167:

```
56  // NEW ADD ACCOUNT_ID_BYTES

73  // NEW ADD pairAccountId

100 // NEW ADD ACCOUNT_ID_BYTES

111         // NEW ADD pairAccountId

123                 // NEW ADD pairAccountId

135         // NEW ADD

141         // NEW ADD

154         // NEW ADD

200 // NEW ADD

209 // NEW ADD

213 // NEW ADD

225 // NEW ADD
```

3.168 CVF-168

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

Recommendation It would be more efficient to pass the initial data offset along with the data, as with the current approach, a caller has to make a slice before calling a "readXXX" function.

Listing 168:

```
61  function readDepositPubdata(bytes memory _data) internal pure
      ↪ returns (Deposit memory parsed) {

104 function readFullExitPubdata(bytes memory _data) internal pure
      ↪ returns (FullExit memory parsed) {

145 function readWithdrawPubdata(bytes memory _data) internal pure
      ↪ returns (Withdraw memory parsed) {

169 function readForcedExitPubdata(bytes memory _data) internal pure
      ↪  returns (ForcedExit memory parsed) {

191 function readChangePubKeyPubdata(bytes memory _data) internal
      ↪ pure returns (ChangePubKey memory parsed) {

214 function readCreatePairPubdata(bytes memory _data) internal pure
      ↪  returns (CreatePair memory parsed)
```

3.169 CVF-169

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

Description These conditions always hold unless there is a bug in the code.
Recommendation Consider using "assert" instead of "require" to emphasize this.

Listing 169:

```
69  require(offset == PACKED_DEPOSIT_PUBDATA_BYTES, "N"); // reading
      ↪  invalid deposit pubdata size

113 require(offset == PACKED_FULL_EXIT_PUBDATA_BYTES, "O"); //
      ↪ reading invalid full exit pubdata size

222 require(offset == PACKED_CREATE_PAIR_PUBDATA_BYTES, "rcp10"); //
      ↪  reading invalid create pair pubdata size
```

3.170 CVF-170

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

Recommendation Probably this constant can be dropped from the encoding.

Listing 170:

```
uint128(0), // amount — ignored
```

3.171 CVF-171

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

Description Unlike some other "readXXX" functions, these functions don't have final offset checks.
Recommendation Consider adding such checks for consistency.

Listing 171:

```
function readWithdrawPubdata(bytes memory _data) internal pure
    ↪ returns (Withdraw memory parsed) {

function readForcedExitPubdata(bytes memory _data) internal pure
    ↪  returns (ForcedExit memory parsed) {

function readChangePubKeyPubdata(bytes memory _data) internal
    ↪ pure returns (ChangePubKey memory parsed) {
```

3.172 CVF-172

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

Recommendation The name 'zkSync' should probably be changed to 'l2swap'

Listing 172:

```
/// @title zkSync configuration constants
```

3.173 CVF-173

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

Description This number looks arbitrary. It is a bad practice to set any hard limits on gas, as gas costs of various operations could change in the future.
Recommendation Consider making this limit configurable.

Listing 173:

```
uint256 constant WITHDRAWAL_GAS_LIMIT = 100000;

uint64 constant MAX_PRIORITY_REQUESTS_TO_DELETE_IN_VERIFY = 6;
```

3.174 CVF-174

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

Description There is no access level specified for these constants so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 174:

```
10 uint256 constant WITHDRAWAL_GAS_LIMIT = 100000;

13 uint8 constant CHUNK_BYTES = 9;

16 uint8 constant ADDRESS_BYTES = 20;

18 uint8 constant PUBKEY_HASH_BYTES = 20;

21 uint8 constant PUBKEY_BYTES = 32;

24 uint8 constant ETH_SIGN_RS_BYTES = 32;

27 uint8 constant SUCCESS_FLAG_BYTES = 1;

30 uint16 constant MAX_AMOUNT_OF_REGISTERED_TOKENS = 127;

33 uint32 constant MAX_ACCOUNT_ID = (2**24) - 1;

36 uint256 constant BLOCK_PERIOD = 15 seconds;

41 uint256 constant EXPECT_VERIFICATION_IN = 0 hours / BLOCK_PERIOD
     ↪ ;

43 uint256 constant NOOP_BYTES = 1 * CHUNK_BYTES;
   uint256 constant DEPOSIT_BYTES = 6 * CHUNK_BYTES;
   uint256 constant TRANSFER_TO_NEW_BYTES = 6 * CHUNK_BYTES;
   uint256 constant WITHDRAW_BYTES = 6 * CHUNK_BYTES;
   uint256 constant TRANSFER_BYTES = 2 * CHUNK_BYTES;
   uint256 constant FORCED_EXIT_BYTES = 6 * CHUNK_BYTES;

50 uint256 constant CREATE_PAIR_BYTES = 4 * CHUNK_BYTES;

53 uint256 constant FULL_EXIT_BYTES = 6 * CHUNK_BYTES;

56 uint256 constant CHANGE_PUBKEY_BYTES = 6 * CHUNK_BYTES;

61 uint256 constant PRIORITY_EXPIRATION_PERIOD = 3 days;
```

(... 64, 69, 72, 75, 79, 82, 86, 89, 92)

3.175 CVF-175

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

Description Using types narrower than 256 bits for constants doesn't increase performance, and actually may even decrease it. Also, it makes the code more error-prone, as arithmetic operations with such constants could be performed in narrowed type terms, thus making overflow more probable.
Recommendation Consider using uint256 type for all the constants with quantity semantics.

Listing 175:

```
13 uint8 constant CHUNK_BYTES = 9;

16 uint8 constant ADDRESS_BYTES = 20;

18 uint8 constant PUBKEY_HASH_BYTES = 20;

21 uint8 constant PUBKEY_BYTES = 32;

24 uint8 constant ETH_SIGN_RS_BYTES = 32;

27 uint8 constant SUCCESS_FLAG_BYTES = 1;

30 uint16 constant MAX_AMOUNT_OF_REGISTERED_TOKENS = 127;

33 uint32 constant MAX_ACCOUNT_ID = (2**24) - 1;

69 uint64 constant MAX_PRIORITY_REQUESTS_TO_DELETE_IN_VERIFY = 6;
```

3.176 CVF-176

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

Description It is a bad practice to rely on any particular block period, as real block periods may be very different.
Recommendation Consider measuring time period in seconds rather than in blocks.

Listing 176:

```
36 uint256 constant BLOCK_PERIOD = 15 seconds;

41 uint256 constant EXPECT_VERIFICATION_IN = 0 hours / BLOCK_PERIOD
    ↪ ;

64 uint256 constant PRIORITY_EXPIRATION =
    ↪ PRIORITY_EXPIRATION_PERIOD / BLOCK_PERIOD;
```

3.177 CVF-177

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

Description A constant always has the same value, so comments like this are confusing.
Recommendation If these constants are actually deployment-time parameters, consider turning them into immutable variables.

Listing 177:

```
/// @dev If set to 0 validator can revert blocks at any time.
uint256 constant EXPECT_VERIFICATION_IN = 0 hours / BLOCK_PERIOD
    ↪ ;

/// @dev NOTE: Priority expiration should be > (
    ↪ EXPECT_VERIFICATION_IN * BLOCK_PERIOD)
/// @dev otherwise incorrect block with priority op could not be
    ↪  reverted.
uint256 constant PRIORITY_EXPIRATION_PERIOD = 3 days;
```

3.178 CVF-178

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

Description This comment in confusing.
Recommendation Consider removing it.

Listing 178:

```
// NEW ADD
```

3.179 CVF-179

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

Description The comment is confusing. As it doesn't explain the semantics of the constant.

Listing 179:

```
/// @dev Timestamp - seconds since unix epoch
uint256 constant COMMIT_TIMESTAMP_NOT_OLDER = 24 hours;
```

3.180 CVF-180

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

Description It is "15 seconds" in the comment and "15 minutes" in the code.
Recommendation Consider fixing the comment.

Listing 180:

```
/// @dev Must be used cause miner's `block.timestamp` value can
    ↪ differ on some small value (as we know — 15 seconds)
uint256 constant COMMIT_TIMESTAMP_APPROXIMATION_DELTA = 15
    ↪ minutes;
```

3.181 CVF-181

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** Config.sol

Description The expression is overcomplicated and unclear.
Recommendation Consider simplifying as: 2**253 - 1.

Listing 181:

```
uint256 constant INPUT_MASK = (~uint256(0) >> 3);
```

3.182 CVF-182

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

Description These functions should be declared as "view".

Listing 182:

```
function getNoticePeriod() external returns (uint256);

function isReadyForUpgrade() external returns (bool);
```

3.183 CVF-183

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

Description The argument name is too long and actually misleading, as it contains upgrade parameters, rather than initialization ones.
Recommendation Consider renaming to "upgradeParameters" or just "parameters".

Listing 183:

```
/// @param newTargetInitializationParameters New target
  ↪ initialization parameters
```

3.184 CVF-184

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** UniswapV2Factory.sol

Recommendation The keys type for this mapping should be “IERC20”. The value type should be “UniswapV2Pair” or “IUniswapV2Pair”.

Listing 184:

```
mapping(address => mapping(address => address)) public getPair;
```

3.185 CVF-185

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** UniswapV2Factory.sol

Recommendation The element type for this array should be “UniswapV2Pair” or “IUniswapV2Pair”.

Listing 185:

```
address[] public allPairs;
```

3.186 CVF-186

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation The type of this variable should be "ZkSync".

Listing 186:

```
address public zkSyncAddress;
```

3.187 CVF-187

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation Events are usually named via nouns, such as "NewPair" or just "Pair".

Listing 187:

```
event PairCreated(address indexed token0, address indexed token1
    ↪ , address pair, uint);
```

3.188 CVF-188

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation The type of the "token0" and "token1" parameters should be "IERC20". The type of the "pair" parameter should be "IUniswapV2Pair".

Listing 188:

```
event PairCreated(address indexed token0, address indexed token1
    ↪ , address pair, uint);
```

3.189 CVF-189

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** UniswapV2Factory.sol

Description The last parameter doesn't have name and the semantics of this parameter is unclear.
Recommendation Consider giving a descriptive name to this parameter.

Listing 189:

```
event PairCreated(address indexed token0, address indexed token1
  ↪ , address pair, uint);
```

3.190 CVF-190

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UniswapV2Factory.sol

Description This constructor is redundant.
Recommendation Consider removing it.

Listing 190:

```
constructor() public {}
```

3.191 CVF-191

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 191:

```
constructor() public {}

function initialize(bytes calldata data) external {}

function upgrade(bytes calldata upgradeParameters) external {}
```

3.192 CVF-192

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation It would be more logical and less error-prone to set the zkSync address in the "initialize" function or even in the constructor. In the latter case, the "zkSyncAddress" variable could be declared as immutable.

Listing 192:

```
function setZkSyncAddress(address _zksyncAddress) external {
```

3.193 CVF-193

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** UniswapV2Factory.sol

Description It is not checked that the new zkSync address is not zero, however zero address value has a special meaning.
Recommendation Consider adding the check.

Listing 193:

```
require(zkSyncAddress == address(0), "szsa1");
zkSyncAddress = _zksyncAddress;
```

3.194 CVF-194

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UniswapV2Factory.sol

Description Once this version is not supposed to be used as an upgrade for any prior version, this function is redundant.
Recommendation Consider removing it or putting "revert()" into its body.

Listing 194:

```
function upgrade(bytes calldata upgradeParameters) external {}
```

3.195 CVF-195

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation The arguments should have type "IERC20". The returned value type should be "IUniswapV2Pair".

Listing 195:

```
function createPair(address tokenA, address tokenB) external
    ↪ returns (address pair) {
```

3.196 CVF-196

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation The same could be done without assembly: new UniswapV2Pair { salt: keccak256 (abi.encodePacked (token0, token1) } ();

Listing 196:

```
bytes memory bytecode = type(UniswapV2Pair).creationCode;
bytes32 salt = keccak256(abi.encodePacked(token0, token1));
assembly {
    pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
}
```

3.197 CVF-197

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** UniswapV2Factory.sol

Description This check is redundant, as it was already checked, that msg.sender == zkSyncAddress, and msg.sender cannot be zero.
Recommendation Consider removing the check or at least moving to the beginning of the function.

Listing 197:

```
require(zkSyncAddress != address(0), 'wzk');
```

3.198 CVF-198

- **Severity** Moderate
- **Category** Bad naming
- **Status** Opened
- **Source** UniswapV2Factory.sol

Description The semantics of the last event parameter is unclear, but the actual value logged here is the index of the just create token pair plus one, which doesn't make much sense.
Recommendation Consider logging the even before pushing the pair into the "allPairs" array.

Listing 198:

```
48 emit PairCreated(token0, token1, pair, allPairs.length);
```

3.199 CVF-199

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation The type of the fist argument should be "IUniswapV2Pair".

Listing 199:

```
51 function mint(address pair, address to, uint256 amount) external
    ↪ {
56 function burn(address pair, address to, uint256 amount) external
    ↪ {
```

3.200 CVF-200

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** UniswapV2Factory.sol

Recommendation The name of the second argument should be "from" for consistency with the "IUniswapV2Pair" interface.

Listing 200:

```
51 function mint(address pair, address to, uint256 amount) external
    ↪ {
56 function burn(address pair, address to, uint256 amount) external
    ↪ {
```

3.201 CVF-201

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** UniswapV2ERC20.sol

Description Mixing the “uint256” type with its alias “uint” in the same file names the code harder to read.
Recommendation Consider using consistent type naming.

Listing 201:

```
using UniswapSafeMath for uint;

event Approval(address indexed owner, address indexed spender, uint256 value);
```

3.202 CVF-202

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** UniswapV2ERC20.sol

Description There is no explicit check that the current balance is sufficient, so the check is actually performed implicitly inside the “sub” function of the “UniswapSafeMath” library. Such implicit checks makes the code harder to read.
Recommendation Consider checking business-level constraints explicitly.

Listing 202:

```
balanceOf[from] = balanceOf[from].sub(value);

balanceOf[from] = balanceOf[from].sub(value);
```

3.203 CVF-203

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** UniswapV2ERC20.sol

Description This logic is non-standard.
Recommendation Consider removing it.

Listing 203:

```
if (allowance[from][msg.sender] != uint256(-1)) {
```

3.204 CVF-204

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UniswapV2ERC20.sol

Description The expression "allowance[from][msg.sender]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 204:

```
if (allowance[from][msg.sender] != uint256(-1)) {
    allowance[from][msg.sender] = allowance[from][msg.sender].
      ↪ sub(value);
```

3.205 CVF-205

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UniswapV2ERC20.sol

Description The expression "allowance[from]" is calculated three times.
Recommendation Consider calculating once and reusing.

Listing 205:

```
if (allowance[from][msg.sender] != uint256(-1)) {
    allowance[from][msg.sender] = allowance[from][msg.sender].
      ↪ sub(value);
```

3.206 CVF-206

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** UQ112x112.sol

Description No access modifier specified for this constant, so the constant will be internal by default.
Recommendation Consider explicitly specifying access level.

Listing 206:

```
uint224 constant Q112 = 2**112;
```

3.207 CVF-207

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UQ112x112.sol

Recommendation Shift would be more efficient that multiplication here.

Listing 207:

```
z = uint224(y) * Q112; // never overflows
```

3.208 CVF-208

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UQ112x112.sol

Description The conversion to the “uint224” type is redundant here.

Listing 208:

```
z = x / uint224(y);
```

3.209 CVF-209

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

Recommendation The type of the “pair” parameter should be “IUniswapV2Pair”.

Listing 209:

```
address pair

address pair
```

3.210 CVF-210

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

Description It is unclear what mapping is meant here.
Recommendation Consider rephrasing.

Listing 210:

```
/// @notice New priority request event. Emitted when a request
    ↪ is placed into mapping
```

3.211 CVF-211

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

Recommendation The word “New” in the event name is redundant.

Listing 211:

```
event NewPriorityRequest(
```

3.212 CVF-212

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

Recommendation These parameters should probably be indexed.

Listing 212:

```
address sender,
uint64 serialId,
Operations.OpType opType,
```

3.213 CVF-213

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

Recommendation This interface should be moved to a seaprate file named “UpgradeEvents”.

Listing 213:

```
interface UpgradeEvents {
```

3.214 CVF-214

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

Recommendation The type of the "upgradeable" parameter should be "Upgradeable".

Listing 214:

```
event NewUpgradable(uint256 indexed versionId, address indexed
    ↪ upgradeable);
```

3.215 CVF-215

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

Description Logging the new targets here is redundant as they were already logged in the corresponding "NoticePeriodStarted" event. What is worth logging are upgrade parameters passed to the new target during upgrade.

Listing 215:

```
event UpgradeComplete(uint256 indexed versionId, address[]
  ↪ newTargets);
```

3.216 CVF-216

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

Description This function is suboptimal.
Recommendation Consider unrolling the loop and using better initial estimation. See, for example, this implementation: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/ABDKMath64x64.sol#L720-L751

Listing 216:

```
function sqrt(uint256 y) internal pure returns (uint256 z) {
```

3.217 CVF-217

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

Description The returned value remains uninitialized in case y is zero.
Recommendation Consider explicitly returning zero in such a case.

Listing 217:

```
function sqrt(uint256 y) internal pure returns (uint256 z) {
```

3.218 CVF-218

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

Recommendation Right shift would be more efficient than division by two.

Listing 218:

```
uint256 x = y / 2 + 1;

      x = (y / x + x) / 2;
```

3.219 CVF-219

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** IUniswapV2Pair.sol

Recommendation Should be "0̂.7.0" for consistency with other files and to follow a common best practice. Unbounded version ranges such as ">=0.5.0" assume that there will be no braking changes in any future version which cannot actually be guaranteed.

Listing 219:

```
pragma solidity >=0.5.0;
```

3.220 CVF-220

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** IUniswapV2Pair.sol

Recommendation These events and functions should be moved to the "IERC20" interface which should be inherited by this interface.

Listing 220:

```
event Approval(address indexed owner, address indexed spender,
  ↪ uint value);
event Transfer(address indexed from, address indexed to, uint
  ↪ value);

function name() external pure returns (string memory);
function symbol() external pure returns (string memory);
function decimals() external pure returns (uint8);
function totalSupply() external view returns (uint);
function balanceOf(address owner) external view returns (uint);
function allowance(address owner, address spender) external view
  ↪  returns (uint);

function approve(address spender, uint value) external returns (
  ↪ bool);
function transfer(address to, uint value) external returns (bool
  ↪ );
function transferFrom(address from, address to, uint value)
  ↪ external returns (bool);
```

3.221 CVF-221

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** IUniswapV2Pair.sol

Recommendation These functions should be moved to "IERC20WithPermit" interface inherited from the "IERC20" interface.

Listing 221:

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function nonces(address owner) external view returns (uint);

function permit(address owner, address spender, uint value, uint
  ↪  deadline, uint8 v, bytes32 r, bytes32 s) external;
```

3.222 CVF-222

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** IUniswapV2Pair.sol

Recommendation This function should be removed from the interface as it is a part of implementation details and is not to be used by third parties.

Listing 222:

```
51 function initialize(address, address) external;
```

3.223 CVF-223

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

Recommendation These values should be calculated only when the “callSuccess” value is true.

Listing 223:

```
34 bool returnedSuccess = callReturnValueEncoded.length == 0 || abi
     ↪ .decode(callReturnValueEncoded, (bool));

55 bool returnedSuccess = callReturnValueEncoded.length == 0 || abi
     ↪ .decode(callReturnValueEncoded, (bool));
```

3.224 CVF-224

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

Recommendation This function could be simplified as: signV := mload(add(_signature, 65)).

Listing 224:

```
76 signV := byte(0, mload(add(_signature, 96)))
```

3.225 CVF-225

- **Severity** Moderate
- **Category** Procedural
- **Status** Opened
- **Source** Utils.sol

Description The "ecrecover" function returns zero address of invalid signature.
Recommendation Add a require statement to check that the returned address is not zero.

Listing 225:

```
return ecrecover(_messageHash, signV, signR, signS);
```

3.226 CVF-226

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** Utils.sol

Description This danages the bytes array passed as argument.
Recommendation Consider restoring the array state after calculating hash, but adding the following line at the end of the assembly block: mstore(_bytes, sub (bytesLen, 32)).

Listing 226:

```
mstore(_bytes, _hash)
```

3.227 CVF-227

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

Recommendation This line could be simplified and made more explicit like this:
return bytes20 (keccak256(_bytes) « 96);

Listing 227:

```
return bytes20(uint160(uint256(keccak256(_bytes))));
```

3.228 CVF-228

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

Description The contract name is inconsistent with the file name.
Recommendation Consider renaming the file to "TokenDeployInit.sol".

Listing 228:

```
contract TokenDeployInit {
```

3.229 CVF-229

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

Description The function name is misleading, as the function always returns an empty array.
Recommendation Consider renaming.

Listing 229:

```
function getTokens() internal pure returns (address[] memory) {
```

3.230 CVF-230

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

Recommendation This variable is redundant as its value is used only once. Just return "new address[](0)".

Listing 230:

```
address[] memory tokens = new address[](0);
```

3.231 CVF-231

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

Recommendation This variable is redundant as it value could be derived from the value of the "upgradePreparationActivationTime".

Listing 231:

```
bool internal upgradePreparationActive;
```

3.232 CVF-232

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

Recommendation This variable should be named "verifierExit" for consistency.

Listing 232:
```
VerifierExit public verifier_exit;
```

3.233 CVF-233

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

Recommendation This variable should be named "pairManager" for consistency.

Listing 233:
```
UniswapV2Factory internal pairmanager;
```

3.234 CVF-234

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

Recommendation In the comment it should be "reserve" instead of "revert".

Listing 234:
```
uint8 internal constant FILLED_GAS_RESERVE_VALUE = 0xff; // we
    ↪ use it to set gas revert value so slot will not be emptied
    ↪  with 0 balance
```

3.235 CVF-235

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

Description This mapping could also use packed account+token pairs as keys, similarly to the "pendingBalances" map.

Listing 235:
```
mapping(uint32 => mapping(uint16 => bool)) public
    ↪ performedExodus;
```

3.236 CVF-236

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

Recommendation It would be more efficient to use a mapping from account IDs to bitmaps of the currency IDs the exodus was performed for. This would require at most 256 words per account to be stored in comparison to 65536 words for the current implementation. Also, the flags for the 128 non-pair tokens would all go into a single word.

Listing 236:

```
54 mapping(uint32 => mapping(uint16 => bool)) public
    ↪ performedExodus;
```

3.237 CVF-237

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

Recommendation Using the "bytes32" and "uint256" types instead of "bytes22" and "uint176" would be more efficient, as Solidity compiler would not need to use shifts and masking.

Listing 237:

```
72 return bytes22((uint176(_address) | (uint176(_tokenId) << 160)))
    ↪ ;
```

3.238 CVF-238

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

Recommendation It would be cheaper to use the "abi.encodePacked" function to pack all the structure fields here. This would allow hashing 140 bytes instead of 192.

Listing 238:

```
93 return keccak256(abi.encode(_storedBlockInfo));
```

3.239 CVF-239

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

Recommendation It could be more efficient to use an array instead of the mapping here.

Listing 239:

```
mapping(uint64 => PriorityOperation) public priorityRequests;
```

3.240 CVF-240

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** SafeMathUInt128.sol

Recommendation These variables could be turned into named return values.

Listing 240:

```
29 uint128 c = a + b;

87 uint128 c = a * b;
```

3.241 CVF-241

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeMathUInt128.sol

Description These variables are redundant.
Recommendation Just return the expression value.

Listing 241:

```
65 uint128 c = a - b;

128 uint128 c = a / b;
```

3.242 CVF-242

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

Recommendation These variables could be turned into named return values.

Listing 242:

```
29  uint256 c = a + b;

87  uint256 c = a * b;
```

3.243 CVF-243

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

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

Listing 243:

```
65  uint256 c = a - b;

128 uint256 c = a / b;
```

3.244 CVF-244

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

Description This line is confusing, as it is unclear a version of what component is referred here.
Recommendation Consider removing this line.

Listing 244:

```
20  * _Available since v2.5.0._
```

3.245 CVF-245

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

Recommendation This function could be optimized like this: function toUint128 (uint256 value) internal pure returns (uint128 result) { require ((result = uint128 (value)) == value, "16"); }

Listing 245:

```
function toUint128(uint256 value) internal pure returns (uint128
  ↪ ) {
```

3.246 CVF-246

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

Recommendation This function could be optimized like this: function toUint64 (uint256 value) internal pure returns (uint64 result) { require ((result = uint64 (value)) == value, "17"); }

Listing 246:

```
function toUint64(uint256 value) internal pure returns (uint64)
  ↪ {
```

3.247 CVF-247

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

Recommendation This function could be optimized like this: function toUint32 (uint256 value) internal pure returns (uint32 result) { require ((result = uint32 (value)) == value, "18"); }

Listing 247:

```
function toUint32(uint256 value) internal pure returns (uint32)
  ↪ {
```

3.248 CVF-248

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

Recommendation This function could be optimized like this: function toUint16 (uint256 value) internal pure returns (uint16 result) { require ((result = uint16 (value)) == value, "19"); }

Listing 248:

```
78 function toUint16(uint256 value) internal pure returns (uint16)
     ↪ {
```

3.249 CVF-249

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

Recommendation This function could be optimized like this: function toUint8 (uint256 value) internal pure returns (uint8 result) { require ((result = uint8 (value)) == value, "1a"); }

Listing 249:

```
93 function toUint8(uint256 value) internal pure returns (uint8) {
```

3.250 CVF-250

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

Recommendation Just initialize the constant with the expression as given in the comment, instead of the hardcoded value. Solidity compiler is smart enough to calculate the constant value at compile time.

Listing 250:

```
 9 /// @dev Storage position of the masters address (keccak256('
     ↪ eip1967.proxy.admin') - 1)
10 bytes32 private constant masterPosition = 0
     ↪ xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103
     ↪ ;
```

3.251 CVF-251

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

Recommendation These variables are redundant. Just use the "masterPosition" constant directly in the assembly blocks.

Listing 251:

```
bytes32 position = masterPosition;

bytes32 position = masterPosition;
```

3.252 CVF-252

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

Description This function should provably emit some event.

Listing 252:

```
function transferMastership(address _newMaster) external {

constructor(address masterAddress) {
```

3.253 CVF-253

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

Recommendation This check is redundant. It is anyway possible to set a dead master address.

Listing 253:

```
require(_newMaster != address(0), "1d"); // otp11 - new masters address can't be zero address
```

3.254 CVF-254

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

Recommendation Just initialize the constant with the expression given in the comment. Solidity compiler is smart enough to calculate such expression at compile time.

Listing 254:

```
uint256 private constant LOCK_FLAG_ADDRESS = 0
    ↪ x8e94fed44239eb2314ab7a406345e6c5a8f0ccedf3b600de3d004e672c33abf4
    ↪ ; // keccak256("ReentrancyGuard") - 1;
```

3.255 CVF-255

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

Description This comment seems to be irrelevant here. Regardless of whether a storage slot is set from non-zero to zero and then back to non-zero, or from non-zero to zero and then back to non-zero, the final outcome is the same: 20K gas used and 15K gas refunded. The benefit of setting the initial value to non-zero appears on on unsuccessful calls, i.e. on reentrancy attempts, as when the initial value is non-zero, we have 15K refund and then revert, while when the initial value is zero, we have 20K gas spent and then revert. Looks at the latest version of the ""ReentrancyGuard"" contract at OpenZeppeling: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol They now have non-zero values for both, entered and non-entered states. The comment is slightly different and a bit confusing but the idea is the following: When you spend 20K gas and then refund 15K gas, the total refund is still limited at 50% of the total transaction cost, so 50% of the total transaction cost is non-refundable. As the total transaction cost is increased by 20K, the non-refundable part is increased by 10K. However, when both entered and non-entered values are non-zero, each write costs only 5K gas, thus the total transaction cost is increased by 10K only and non-refundable part is increased by 5K.

Listing 255:

```
// Storing an initial non-zero value makes deployment a bit more
// expensive, but in exchange the refund on every call to
    ↪ nonReentrant
// will be lower in amount. Since refunds are capped to a
    ↪ percetange of
// the total transaction's gas, it is best to keep them low in
    ↪ cases
// like this one, to increase the likelihood of the full refund
    ↪ coming
// into effect.
```

3.256 CVF-256

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

Recommendation Consider giving descriptive names to the "0" and "1" constants here, like "ENTERED" and "NOT_ENTERED".

Listing 256:

```
sstore(LOCK_FLAG_ADDRESS, 1)

sstore(LOCK_FLAG_ADDRESS, 0)

sstore(LOCK_FLAG_ADDRESS, 1)
```

3.257 CVF-257

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

Recommendation This function could be simplified like this: assembly { _bts := mload (0x40) mstore (0x40, add (_bts, 0x22)) mstore (add (_bts, 2), self) mstore (_bts, 2) } This would make the function about three times more efficient.

Listing 257:

```
function toBytesFromUInt16(uint16 self) internal pure returns (
    ↪ bytes memory _bts) {
```

3.258 CVF-258

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

Recommendation "This function could be simplified like this: assembly { _bts := mload (0x40) mstore (0x40, add (_bts, 0x23)) mstore (add (_bts, 3), self) mstore (_bts, 3) } This would make the function about three times more efficient."

Listing 258:

```
function toBytesFromUInt24(uint24 self) internal pure returns (
    ↪ bytes memory _bts) {
```

3.259 CVF-259

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

Recommendation "This function could be simplified like this: assembly { _bts := mload (0x40) mstore (0x40, add (_bts, 0x24)) mstore (add (_bts, 4), self) mstore (_bts, 4) } This would make the function about three times more efficient."

Listing 259:

```
22 function toBytesFromUInt32(uint32 self) internal pure returns (
    ↪ bytes memory _bts) {
```

3.260 CVF-260

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

Recommendation "This function could be simplified like this: assembly { _bts := mload (0x40) mstore (0x40, add (_bts, 0x30)) mstore (add (_bts, 16), self) mstore (_bts, 16) } This would make the function about three times more efficient."

Listing 260:

```
26 function toBytesFromUInt128(uint128 self) internal pure returns
    ↪ (bytes memory _bts) {
```

3.261 CVF-261

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

Recommendation Shift would be more efficient than multiplication here.

Listing 261:

```
uint256 data = self << ((32 - byteLength) * 8);

return a >> ((0x20 - _new_length) * 8);

return bytes1(uint8(0x66656463626139383736353433323130 >> (uint8
    ↪ (_byte) * 8)));

bytes memory outStringBytes = new bytes(_input.length * 2);

      // outStringByte = byte (uint8 (0
        ↪ x66656463626139383736353433323130 >> (uint8 (_byteHalf
        ↪ ) * 8)))

            shl(0xf8, shr(mul(shr(0x04, curr_input_byte), 0x08), 0
              ↪ x66656463626139383736353433323130))

            shl(0xf8, shr(mul(and(0x0f, curr_input_byte), 0x08), 0
              ↪ x66656463626139383736353433323130))
```

3.262 CVF-262

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

Recommendation "This function could be simplified like this: assembly { _bts := mload (0x40) mstore (0x40, add (_bts, 0x34)) mstore (add (_bts, 20), self) mstore (_bts, 20) } This would make the function about three times more efficient."

Listing 262:

```
function toBytesFromAddress(address self) internal pure returns
    ↪ (bytes memory bts) {
```

3.263 CVF-263

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

Description The names of these functions are confusing, as they don't convert the whole bytes object to the destination type but rather extract a value of the destination type from the middle of the bytes object.
Recommendation Consider renaming.

Listing 263:

```
function bytesToAddress(bytes memory self, uint256 _start) internal pure returns (address addr) {

function bytesToBytes20(bytes memory self, uint256 _start) internal pure returns (bytes20 r) {

function bytesToUInt16(bytes memory _bytes, uint256 _start) internal pure returns (uint16 r) {

function bytesToUInt24(bytes memory _bytes, uint256 _start) internal pure returns (uint24 r) {

function bytesToUInt32(bytes memory _bytes, uint256 _start) internal pure returns (uint32 r) {

function bytesToUInt128(bytes memory _bytes, uint256 _start) internal pure returns (uint128 r) {

function bytesToUInt160(bytes memory _bytes, uint256 _start) internal pure returns (uint160 r) {

function bytesToBytes32(bytes memory _bytes, uint256 _start) internal pure returns (bytes32 r) {
```

3.264 CVF-264

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

Recommendation Consider explaining why overflow is not possible in practice or add an overflow protection.

Listing 264:

```
// NOTE: theoretically possible overflow of (_start + 20)
// NOTE: theoretically possible overflow of (_start + 0x2)
// NOTE: theoretically possible overflow of (_start + 0x3)
// NOTE: theoretically possible overflow of (_start + 0x4)
// NOTE: theoretically possible overflow of (_start + 0x10)
// NOTE: theoretically possible overflow of (_start + 0x14)
// NOTE: theoretically possible overflow of (_start + 0x20)
// NOTE: theoretically possible overflow of (_start + _length)
// NOTE: theoretically possible overflow of (_offset + 1)
// NOTE: theoretically possible overflow of (_offset + 1)
// NOTE: theoretically possible overflow of (_offset + 2)
// NOTE: theoretically possible overflow of (_offset + 3)
// NOTE: theoretically possible overflow of (_offset + 4)
// NOTE: theoretically possible overflow of (_offset + 16)
// NOTE: theoretically possible overflow of (_offset + 20)
// NOTE: theoretically possible overflow of (_offset + 20)
// NOTE: theoretically possible overflow of (_offset + 20)
// NOTE: theoretically possible overflow of (_offset + 32)
```

3.265 CVF-265

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

Description The expression "_start + 20" is calculated twice: once in Solidity and another time in assembly.
Recommendation Consider calculating once and reusing.

Listing 265:

```
require(self.length >= (_start + 20), "S");

    r := mload(add(add(self, 0x20), _start))
```

3.266 CVF-266

- **Severity** Major
- **Category** Suboptimal
- **Status** Info
- **Source** Bytes.sol

Recommendation This may read beyond the source array and write beyond the destination array. Probably not a major issue, but could be fixed quite easily: assembly { result := mload (0x40) mstore (0x40, add (result, add (0x20, _length)))
for { let fromPtr := add (_bytes, add (_start, _length)) let toPtr := add (result, _length) } lt (result, toPtr) { fromPtr := sub (fromPtr, 0x20) toPtr := sub (toPtr, 0x20) } { mstore (toPtr, mload (fromPtr)) }
mstore (result, _length) }
Client Comment The caller 'collectOnChainOps' ensure the correct argument.

Listing 266:

```
// NOTE: theoretically possible overflow of (_start + _length)
```

3.267 CVF-267

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

Recommendation This function could be significantly optimized by converting 16 bytes at once. The following gist illustrates the idea: https://gist.github.com/3sGgpQ8H/7c00e48af22b91effd5c37adc44908f3

Listing 267:

```
function bytesToHexASCIIBytes(bytes memory _input) internal pure
    ↪ returns (bytes memory _output) {
```

3.268 CVF-268

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

Description This function should return string instead of bytes.

Listing 268:

```
function bytesToHexASCIIBytes(bytes memory _input) internal pure returns (bytes memory _output) {
```

3.269 CVF-269

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

Recommendation The left shift wouldn't be necessary in case the “MSTORE8” opcode would be used instead of “MSTORE”.

Listing 269:

```
mstore(

shl(0xf8, shr(mul(shr(0x04, curr_input_byte), 0x08), 0x66656463626139383736353433323130))

mstore(

shl(0xf8, shr(mul(and(0x0f, curr_input_byte), 0x08), 0x66656463626139383736353433323130))
```