ABDK CONSULTING

SMART CONTRACT AUDIT

ZkSync

Version 8

Solidity

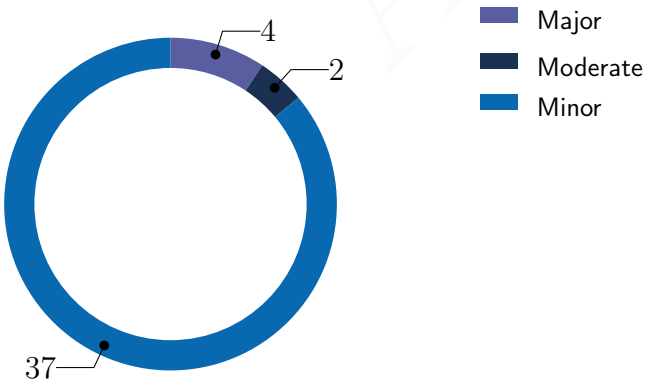
abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
29th March 2022

We've been asked to review the 19 files in a new version at tag contracts-8. We found 4 major, and a few less important issues.
All identified major issues have been fixed or otherwise addressed in collaboration with the client. These contracts were deployed:

- Governance
- ZkSync



Findings

ID	Severity	Category	Status
CVF-1	Minor	Readability	Info
CVF-2	Major	Flaw	Info
CVF-3	Minor	Suboptimal	Fixed
CVF-4	Minor	Suboptimal	Info
CVF-5	Minor	Procedural	Fixed
CVF-6	Minor	Suboptimal	Info
CVF-7	Moderate	Flaw	Fixed
CVF-8	Minor	Suboptimal	Fixed
CVF-9	Minor	Bad naming	Fixed
CVF-10	Minor	Bad naming	Fixed
CVF-11	Minor	Suboptimal	Info
CVF-12	Moderate	Bad datatype	Info
CVF-13	Minor	Procedural	Fixed
CVF-14	Minor	Procedural	Fixed
CVF-15	Minor	Procedural	Info
CVF-16	Minor	Suboptimal	Info
CVF-17	Minor	Procedural	Info
CVF-18	Minor	Unclear behavior	Fixed
CVF-19	Minor	Readability	Fixed
CVF-20	Minor	Procedural	Fixed
CVF-21	Minor	Readability	Info
CVF-22	Minor	Readability	Info
CVF-23	Minor	Unclear behavior	Fixed
CVF-24	Minor	Readability	Info
CVF-25	Minor	Procedural	Fixed
CVF-26	Minor	Readability	Info
CVF-27	Minor	Bad datatype	Info

ID	Severity	Category	Status
CVF-28	Minor	Readability	Info
CVF-29	Minor	Readability	Info
CVF-30	Minor	Readability	Fixed
CVF-31	Minor	Flaw	Fixed
CVF-32	Minor	Unclear behavior	Info
CVF-33	Minor	Suboptimal	Fixed
CVF-34	Major	Flaw	Fixed
CVF-35	Minor	Documentation	Fixed
CVF-36	Minor	Bad naming	Fixed
CVF-37	Major	Flaw	Info
CVF-38	Minor	Suboptimal	Fixed
CVF-39	Minor	Procedural	Fixed
CVF-40	Minor	Bad naming	Fixed
CVF-41	Major	Suboptimal	Info
CVF-42	Minor	Suboptimal	Info
CVF-43	Minor	Suboptimal	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	March 28, 2022	D. Khovratovich	Initial Draft
0.2	March 28, 2022	D. Khovratovich	Minor revision
1.0	March 28, 2022	D. Khovratovich	Release
1.1	March 29, 2022	D. Khovratovich	Client comments update
1.2	March 29, 2022	D. Khovratovich	Date update
1.3	March 29, 2022	D. Khovratovich	Listing bug fix
1.4	March 29, 2022	D. Khovratovich	CVF-17, link added for readability
2.0	March 29, 2022	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 the repository:

- AdditionalZkSync.sol
- Config.sol
- Create2Factory.sol
- DeployFactory.sol
- Events.sol
- Governance.sol
- IERC20.sol
- Operations.sol
- PlonkCore.sol
- Proxy.sol
- ReentrancyGuard.sol
- RegenesisMultisig.sol
- Storage.sol
- TokenGovernance.sol
- UpgradeGatekeeper.sol
- Utils.sol
- Verifier.sol
- ZkSync.sol
- ZkSyncNFTFactory.sol

The fixes were provided in a new pull request.

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
- **Status** Info
- **Category** Readability
- **Source** AdditionalZkSync.sol

Description These changes only affect code formatting.
Recommendation Consider reverting them to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 1:

```
61 -bool proofCorrect =
   -     verifier.verifyExitProof(
   -         _storedBlockInfo.stateHash,
   -         _accountId,
   -         _owner,
   -         _tokenId,
   -         _amount,
   -         _nftCreatorAccountId,
   -         _nftCreatorAddress,
70 -         _nftSerialId,
   -         _nftContentHash,
   -         _proof
   -     );
   +bool proofCorrect = verifier.verifyExitProof(
   +     _storedBlockInfo.stateHash,
   +     _accountId,
   +     _owner,
   +     _tokenId,
   +     _amount,
80 +     _nftCreatorAccountId,
   +     _nftCreatorAddress,
   +     _nftSerialId,
   +     _nftContentHash,
   +     _proof
   +);
```

(94, 100)

3.2 CVF-2

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

Description The conversion to "uint16" is safe here only because the current "MAX_FUNGIBLE_TOKEN_ID" value fits into 16 bits, while the "MAX_FUNGIBLE_TOKEN_ID" constant has type "uint32" and thus allows higher values. Code correctness shouldn't depend on particular values of configuration parameters.
Recommendation Consider removing the conversion and changing the corresponding event parameter type to "uint32".
Client Comment It's made for consistency, 'packAddressAndTokenId', 'increaseBalanceToWithdraw', already used 'uint16' for token ID, it can be changed but with a big legacy or more inconsistency

Listing 2:

```
91 +emit WithdrawalPending(uint16(_tokenId), _owner, _amount);
```

3.3 CVF-3

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AdditionalZkSync.sol

Recommendation The function should break from the loop in case the first part of the conjunction is true, regardless of the second part.
Client Comment -

Listing 3:

```
159 +if (SECURITY_COUNCIL_MEMBERS[id] == addr && !
    ↪ securityCouncilApproves[id]) {
```

3.4 CVF-4

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AdditionalZkSync.sol

Recommendation As member IDs go in sequence starting from zero, a bit mask would be more efficient than a mapping. Also, using a bit mask would make the "numberOfApprovalsFromSecurityCouncil" counter redundant, as the number of set bits in a bit mask could be calculated quite cheaply.
Client Comment Readability is preferable than gas optimisation in function that to be called only in cases of emergency

Listing 4:

```
160 securityCouncilApproves[id] = true;
```

3.5 CVF-5

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** AdditionalZkSync.sol

Recommendation This check should be performed inside the "recoverAddressFromEthSignature" function.
Client Comment -

Listing 5:

```
219 +               require(recoveredAddress != address(0x00), ""p4"");
    ↪ // invalid signature"
```

3.6 CVF-6

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** AdditionalZkSync.sol

Description Here the same contract is invoked three times.
Recommendation It would be more efficient to implement a function in the gate keeper contract that returns several addresses at once.
Client Comment It is totally makes sense, but for do it we should change owner of all proxies to new upgradeGatekeeper. It can be added in the next upgrades

Listing 6:

```
229 +(, bytes memory newTarget0) = gatekeeper.staticcall(abi.
    ↪ encodeWithSignature("nextTargets(uint256)", 0));
230 +(, bytes memory newTarget1) = gatekeeper.staticcall(abi.
    ↪ encodeWithSignature("nextTargets(uint256)", 1));
+(, bytes memory newTarget2) = gatekeeper.staticcall(abi.
    ↪ encodeWithSignature("nextTargets(uint256)", 2));
```

3.7 CVF-7

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** AdditionalZkSync.sol

Description The execution status of the invocations is ignored, so unsuccessful invocations are treated as if they returned empty byte strings. It is possible to make an internal contract invocation unsuccessful by manipulating the transaction gas limit.
Recommendation Consider reverting on unsuccessful invocations.
Client Comment -

Listing 7:

```
229 +(, bytes memory newTarget0) = gatekeeper.staticcall(abi.
    ↪ encodeWithSignature("nextTargets(uint256)", 0));
230 +(, bytes memory newTarget1) = gatekeeper.staticcall(abi.
    ↪ encodeWithSignature("nextTargets(uint256)", 1));
+(, bytes memory newTarget2) = gatekeeper.staticcall(abi.
    ↪ encodeWithSignature("nextTargets(uint256)", 2));
```

3.8 CVF-8

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

Recommendation Consider using uint32 type for all token index variables and constants.
Client Comment -

Listing 8:

```
32 uint16 internal constant MAX_AMOUNT_OF_REGISTERED_TOKENS = $(
    ↪ MAX_AMOUNT_OF_REGISTERED_TOKENS);
```

3.9 CVF-9

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** Create2Factory.sol

Description This argument is a bytecode, not a bytecode hash.
Recommendation Consider renaming.
Client Comment -

Listing 9:

```
33 +bytes memory bytecodeHash,
```

3.10 CVF-10

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** Create2Factory.sol

Description This variable contains the keccak256 hash of a bytecode, not a hash of a hash of a bytecode.
Recommendation Consider renaming.
Client Comment -

Listing 10:

```
36 +bytes32 bytecodeHashHash = keccak256(bytecodeHash);
```

3.11 CVF-11

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

Description This change only affects code formatting.
Recommendation Consider reverting it to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 11:

```
62 -Proxy zkSync =
-     new Proxy(
-         address(_zksyncTarget),
-         abi.encode(address(governance), address(verifier),
    ↪ address(additionalZkSync), _genesisRoot)
-     );
+Proxy zkSync = new Proxy(
+     address(_zksyncTarget),
+     abi.encode(address(governance), address(verifier), address(
    ↪ additionalZkSync), _genesisRoot)
70 +);
```

3.12 CVF-12

- **Severity** Moderate
- **Status** Info
- **Category** Bad datatype
- **Source** Events.sol

Recommendation The type of the "tokenId" parameter should be "uint32" as token IDs are 32 bits in general. It is true that only a fungible token ID may appear here and a fungible token ID always fits into 16 bits, but this fact is true only because of the particular value of a configuration constant, and correctness of the code shouldn't depend on configuration constant values.
Client Comment It's made for consistency, 'packAddressAndTokenId', 'increaseBalanceToWithdraw', already used 'uint16' for token ID, it can be changed but at the end with a big legacy

Listing 12:

```
19  event Withdrawal(uint16 indexed tokenId, uint128 amount);

23 +event WithdrawalPending(uint16 indexed tokenId, address
    ↪ recepient, uint128 amount);
```

3.13 CVF-13

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

Recommendation The "recipient" parameter should be indexed.
Client Comment -

Listing 13:

```
23 +event WithdrawalPending(uint16 indexed tokenId, address
   ↪ recepient, uint128 amount);
```

3.14 CVF-14

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

Recommendation The "addr" parameter should be indexed.
Client Comment -

Listing 14:

```
71 +event ApproveCutUpgradeNoticePeriod(address addr);
```

3.15 CVF-15

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

Recommendation Consider using uint32 type universally for all token index variables.
Client Comment It's made for consistency, 'packAddressAndTokenId', 'increaseBalanceToWithdraw', already used 'uint16' for token ID, it can be changed but at the end with a big legacy

Listing 15:

```
41  uint16 public totalTokens;

44  mapping(uint16 => address) public tokenAddresses;

47  mapping(address => uint16) public tokenIds;

53  mapping(uint16 => bool) public pausedTokens;

106     uint16 newTokenId = totalTokens; // it is not 'totalTokens -
            ↪ 1' because tokenId = 0 is reserved for eth

119     uint16 tokenId = this.validateTokenAddress(_tokenAddr);

160     uint16 tokenId = tokenIds[_tokenAddr];
```

3.16 CVF-16

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

Recommendation It would be cheaper to use the "EXTCODEHASH" opcode here and compare its output with two values: zero and the keccak256 hash of an empty string. See ERC-1052 for details: https://eips.ethereum.org/EIPS/eip-1052
Client Comment 'extcodesize' will be even cheaper, because it has to be compared with only one value and not with two, and opcodes cost the same

Listing 16:

```
229 +contractSize := extcodesize(_address)
```

3.17 CVF-17

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IERC20.sol

Description Why is that done? The resulting methods are not compliant with EIP20.
Client Comment Actually, some ERC20s incorrectly implement the standard, but remain in use (see the next link). We decided to use interfaces for ‘transfer‘/‘transferFrom‘ as it is cheaper due to serialization

Listing 17:

```
28 +function transfer(address recipient, uint256 amount) external;

69 +) external;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IERC20.sol

Description The standard is not strict enough to tell for sure what implementations do exactly match it. The standard allows the “transfer” and “transferFrom“ functions to either revert of return false un failure and requires them to return true on success. it is unclear whether this interface allows reverting on unsuccessful transfer attempts or requires an implementation to explicitly return false.
Recommendation Consider clarifying and/or giving examples of behaviors that are assumed exactly matching the standard and not exactly matching it.
Client Comment -

Listing 18:

```
89 +/// Note: It is assumed that the interface applies to those ‘
     ↪ ERC20‘ tokens whose code exactly matches the standard.
90 +/// Note: Used to perform transfers for tokens that explicitly
     ↪ return a boolean value
```

3.19 CVF-19

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** IERC20.sol

Recommendation Should be "will revert" or "will be reverted".
Client Comment -

Listing 19:

```
91 +/// (if the token returns any other data or does not return at
   ↪ all, then the function call will reverted)
```

3.20 CVF-20

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IERC20.sol

Recommendation This interface should be defined in a separate file named "ITrustedTransferableERC20.sol".
Client Comment -

Listing 20:

```
92 +interface ITrustedTransfarableERC20 {
```

3.21 CVF-21

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** PlonkCore.sol

Description These changes affect only code formatting.
Recommendation Consider reverting them to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 21:

```
984 -(uint256 recursive_input, PairingsBn254.G1Point[2] memory
      ↪ aggregated_g1s) =
-     reconstruct_recursive_public_input(
-         recursive_vks_root,
-         max_valid_index,
-         recursive_vks_indexes,
-         individual_vks_inputs,
990 -         subproofs_limbs
-     );
+(uint256 recursive_input, PairingsBn254.G1Point[2] memory
      ↪ aggregated_g1s) = reconstruct_recursive_public_input(
+     recursive_vks_root,
+     max_valid_index,
+     recursive_vks_indexes,
+     individual_vks_inputs,
+     subproofs_limbs
+);

1198 -bool valid =
-     verify_recursive(
1200 -         proof,
-         vk,
-         recursive_vks_root,
-         max_valid_index,
-         recursive_vks_indexes,
-         individual_vks_inputs,
-         subproofs_limbs
-     );
+bool valid = verify_recursive(
+     proof,
1210 +     vk,
+     recursive_vks_root,
+     max_valid_index,
+     recursive_vks_indexes,
+     individual_vks_inputs,
+     subproofs_limbs
```

3.22 CVF-22

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** Proxy.sol

Description These changes affect only code formatting.
Recommendation Consider reverting them to reduce code size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 22:

```
22 -(bool initializationSuccess, ) =
-     getTarget().delegatecall(abi.encodeWithSignature("
   ↪ initialize(bytes)", targetInitializationParameters));
+(bool initializationSuccess, ) = getTarget().delegatecall(
+     abi.encodeWithSignature("initialize(bytes)",
   ↪ targetInitializationParameters)
+);

65 -(bool upgradeSuccess, ) =
-     getTarget().delegatecall(abi.encodeWithSignature("upgrade(
   ↪ bytes)", newTargetUpgradeParameters));
+(bool upgradeSuccess, ) = getTarget().delegatecall(
+     abi.encodeWithSignature("upgrade(bytes)",
   ↪ newTargetUpgradeParameters)
+);

91 -         case 0 {
-             // End execution and revert state changes
-             revert(ptr, size)
-         }
-         default {
-             // Return data with length of size at pointers
   ↪ position
-             return(ptr, size)
-         }
+     case 0 {
100 +         // End execution and revert state changes
+         revert(ptr, size)
+     }
+     default {
+         // Return data with length of size at pointers position
+         return(ptr, size)
+     }
```

3.23 CVF-23

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** ReentrancyGuard.sol

Description The fact the some slot is empty doesn't necessary mean that it is not in use, so this check doesn't completely eliminates possibility of slot conflict. What this check actually prevents is possibility for double initialization.
Client Comment -

Listing 23:

```
55 +// Check that storage slot for reentrancy guard is empty to
       ↪ rule out possibility of slot conflict
   +require(lockSlotOldValue == 0, "1B");
```

3.24 CVF-24

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** RegenesisMultisig.sol

Description This change affects only code formatting.
Recommendation Consider reverting it to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 24:

```
52 -address payable[SECURITY_COUNCIL_MEMBERS_NUMBER] memory
       ↪ SECURITY_COUNCIL_MEMBERS =
   -     [$(SECURITY_COUNCIL_MEMBERS)];
   +address payable[SECURITY_COUNCIL_MEMBERS_NUMBER] memory
       ↪ SECURITY_COUNCIL_MEMBERS = [
   +     $(SECURITY_COUNCIL_MEMBERS)
   +];
```

3.25 CVF-25

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** TokenGovernance.sol

Recommendation This event should include the new listing fee amount as a parameter.
Client Comment -

Listing 25:

```
20 +event ListingFeeTokenUpdate(ITrustedTransfarableERC20 indexed
   ↪ newListingFeeToken);
```

3.26 CVF-26

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** UpgradeGatekeeper.sol

Description This change only affects code formatting.
Recommendation Consider reverting it to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 26:

```
21 -enum UpgradeStatus {Idle, NoticePeriod, Preparation}
   +enum UpgradeStatus {
   +    Idle,
   +    NoticePeriod,
   +    Preparation
   +}
```

3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Utils.sol

Recommendation A “unt256” version would be more useful and probably a bit more efficient.
Client Comment Using one function with "uint256" is a bit uncomfortable, because need to convert all inputs to ‘uint256‘

Listing 27:

```
58 +function minU128(uint128 a, uint128 b) internal pure returns (
   ↪ uint128) {
```

3.28 CVF-28

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

Description This change only affects code formatting.
Recommendation Consider reverting it to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 28:

```
69 -bytes32 commitment =
70 -     sha256(
   -         abi.encodePacked(
   -             _rootHash,
   -             _accountId,
   -             _owner,
   -             _tokenId,
   -             _amount,
   -             _nftCreatorAccountId,
   -             _nftCreatorAddress,
   -             _nftSerialId,
80 -             _nftContentHash
   -         )
   -     );
   +bytes32 commitment = sha256(
   +     abi.encodePacked(
   +         _rootHash,
   +         _accountId,
   +         _owner,
   +         _tokenId,
   +         _amount,
90 +         _nftCreatorAccountId,
   +         _nftCreatorAddress,
   +         _nftSerialId,
   +         _nftContentHash
   +     )
   +);
```

3.29 CVF-29

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** ZkSync.sol

Description These changes only affect code formatting.
Recommendation Consider reverting them to reduce the diff size.
Client Comment At some point we updated the lint, this will not affect the diff of subsequent audits, so now we see no reason to roll it back

Listing 29:

```
148 -(address _governanceAddress, address _verifierAddress, address
    ↪ _additionalZkSync, bytes32 _genesisStateHash) =
-     abi.decode(initializationParameters, (address, address,
    ↪ address, bytes32));
150 +(
+     address _governanceAddress,
+     address _verifierAddress,
+     address _additionalZkSync,
+     bytes32 _genesisStateHash
+) = abi.decode(initializationParameters, (address, address,
    ↪ address, bytes32));

161 -StoredBlockInfo memory storedBlockZero =
-     StoredBlockInfo(0, 0, EMPTY_STRING_KECCAK, 0,
    ↪ _genesisStateHash, bytes32(0));
-
+StoredBlockInfo memory storedBlockZero = StoredBlockInfo(
+     0,
+     0,
+     EMPTY_STRING_KECCAK,
+     0,
+     _genesisStateHash,
170 +     bytes32(0)
+);

401 -Operations.FullExit memory op =
-     Operations.FullExit({
-         accountId: _accountId,
-         owner: msg.sender,
-         tokenId: tokenId,
-         amount: 0, // unknown at this point
-         nftCreatorAccountId: uint32(0), // unknown at this
    ↪ point
-         nftCreatorAddress: address(0), // unknown at this point
-         nftSerialId: uint32(0), // unknown at this point
```

(..., 410, 441, 486, 675,685,747,855,1001,1057,1110,1151,1247)

3.30 CVF-30

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** ZkSync.sol

Recommendation Should be "if the balance of the contract decreased after transfer".
Client Comment -

Listing 30:

```
240 +require(balanceDiff > 0, "c1"); // transfer is considered
    ↪ successful only if the balance of the contract increased
    ↪ after transfer
```

3.31 CVF-31

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

Description This is true for the mainnet, while in test networks and Ethereum clones this statement could be false.
Recommendation Consider using a checked conversion.
Client Comment -

Listing 31:

```
267 +// - 1 Ether is 10^18 Wei
    +// - Total supply of Ether is 118,019,446 (as of December 27,
    ↪ 2021)
    +// - 2^128 > 10^38 > (Total supply of Ether) * 10^18
```

3.32 CVF-32

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** ZkSync.sol

Recommendation This assignment should be performed in the "else" branch of the conditional statement below. Otherwise 'tokenId' is assigned with a return value from 'validateTokenAddress'.

Listing 32:

```
325 +uint16 tokenId = 0;
```

3.33 CVF-33

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

Description The "pendingBalances[pacjedBalanceKe].balanceToWithdraw" value is ponentially updated twice.
Recommendation Consider refactoring the code to update it at most once.
Client Comment -

Listing 33:

```
336 +pendingBalances[packedBalanceKey].balanceToWithdraw = balance -
    ↪  amount;

355 +           pendingBalances[packedBalanceKey].balanceToWithdraw =
    ↪ balance - withdrawnAmount;
```

3.34 CVF-34

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

Recommendation The second parameter should be the real withdrawn amount, that could be different from the "_amount" value.
Client Comment -

Listing 34:

```
358 +emit Withdrawal(tokenId, amount);
```

3.35 CVF-35

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

Description This function doesn't seem to actually emit events. Solidity wouldn't allow to declare it as "view" in case it would emit any events.
Client Comment -

Listing 35:

```
469 +/// @dev NOTE: does not change storage (only emit events)!
```

3.36 CVF-36

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

Description The function name is confusing, as it actually increases the pending balance by the given amount instead of overwriting it.
Client Comment -

Listing 36:

```
598 +function storePendingBalance(
```

3.37 CVF-37

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

Description The fact that a fungible token ID always fits into 16 bits is based on the actual value of a configuration parameter. Code correctness shouldn't depend on particular values of configuration parameters.
Recommendation Consider using the "uint32" type for token IDs or using "type(uint16).max" instead of the "MAX_FUNGIBLE_TOKEN_ID" constant.
Client Comment Using "uint32" can be very inconvenient now because packed balance mapping needs to be rewritten, and using "type(uint16).max" instead of "MAX_FUNGIBLE_TOKEN_ID" can be confusing. It can be changed but at the end with a big legacy. Maybe it is better to leave it as is.

Listing 37:

```
661 +handleWithdrawFT(_completeWithdrawals, uint16(op.tokenId), op.
    ↪ owner, op.amount);

667 +handleWithdrawFT(_completeWithdrawals, uint16(op.tokenId), op.
    ↪ target, op.amount);

672 +    handleWithdrawFT(_completeWithdrawals, uint16(op.tokenId),
    ↪ op.owner, op.amount);
```

3.38 CVF-38

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

Description If the contract is already in exodus mode, the function reverts, while in other circumstances, that prevent entering exodus mode, the function returns false. It s a bad practice to combine in a single function different ways of error reporting.
Recommendation Consider returning false (or true) in case exodus mode is already active.
Client Comment -

Listing 38:

```
782 +requireActive();
```

3.39 CVF-39

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

Recommendation The check for zero recovered address should be moved to the "recoverAddressFromEthSignature" function.
Client Comment -

Listing 39:

```
1021 +return recoveredAddress == _changePk.owner && recoveredAddress
     ↪ != address(0);
```

3.40 CVF-40

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** ZkSyncNFTFactory.sol

Recommendation Constants are usually named IN_UPPER_CASE.
Client Comment -

Listing 40:

```
17 +bytes constant sha256MultiHash = hex"1220";
```

3.41 CVF-41

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

Description Inputs of this function are known to always be 34 bytes long. This allows significantly optimizing the function by treating the input as an 272-bits integer and converting it into base58 representation using division and modulo operations. The first few iterations will require dealing with numbers longer than 256 bits, but such operations are still quite cheap in ethereum. The vast majority of iterations would deal with numbers that already fit into 256 bits.
Client Comment These functions are 'pure' and 'view' and it's like no one uses onchain in a transaction, so that such an optimization will only make the code harder to understand

Listing 41:

```
135 +function toBase58(bytes memory source) internal pure returns (
    ↪ string memory) {
```

3.42 CVF-42

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ZkSyncNFTFactory.sol

Description This function copies bytes one by one, which is inefficient.
Recommendation Consider optimizing using the following function that reverses bytes in a 32-bytes word: function reverseBytes (uint x) public pure returns (uint) { x = x « 128 | x » 128; x = (x & 0xFFFFFFFFFFFFFFFF0000000000000000FFFFFFFFFFFFFFFF) « 64 | x » 64 & 0xFFFFFFFFFFFFFFFF0000000000000000FFFFFFFFFFFFFFFF; x = (x & 0xFFFFFFFF00000000FFFFFFFF00000000FFFFFFFF00000000FFFFFFFF) « 32 | x » 32 & 0xFFFFFFFF00000000FFFFFFFF00000000FFFFFFFF00000000FFFFFFFF; x = (x & 0xFFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF) « 16 | x » 16 & 0xFFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF; return (x & 0xFF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF) « 8 | x » 8 & 0xFF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF; }
Client Comment These functions are 'pure' and 'view' and it's like no one uses onchain in a transaction, so that such an optimization will only make the code harder to understand

Listing 42:

```
160 +function reverse(uint8[] memory input) internal pure returns (
    ↪ uint8[] memory) {
```

3.43 CVF-43

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ZkSyncNFTFactory.sol

Description This function converts bytes one by one which is inefficient.
Recommendation Consider optimizing using the following, that converts 32 bytes at once: function toAlphabet (uint x) public pure returns (uint) { x += x + 0x5454545454545454545454545454545454545454545454545454545454545454 » 7 & 0x0101010101010101010101010101010101010101010101010101010101010101; x += (x + 0x5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F5F) » 7 & 0x0101010101010101010101010101010101010101010101010101010101010101) * 6; x += x + 0x6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A » 7 & 0x0101010101010101010101010101010101010101010101010101010101010101; x += x + 0x6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F6F » 7 & 0x0101010101010101010101010101010101010101010101010101010101010101; x += (x + 0x7777777777777777777777777777777777777777777777777777777777777777 » 7 & 0x0101010101010101010101010101010101010101010101010101010101010101) * 7; return x + 0x3131313131313131313131313131313131313131313131313131313131313131; }
Client Comment These functions are ‘pure‘ and ‘view‘ and it’s like no one uses onchain in a transaction, so that such an optimization will only make the code harder to understand

Listing 43:

```
168 +function toAlphabet(uint8[] memory indices) internal pure
    ↪ returns (string memory) {
```