ABDK CONSULTING

SMART CONTRACT
AUDIT

Matter Labs

ZkSync. V9

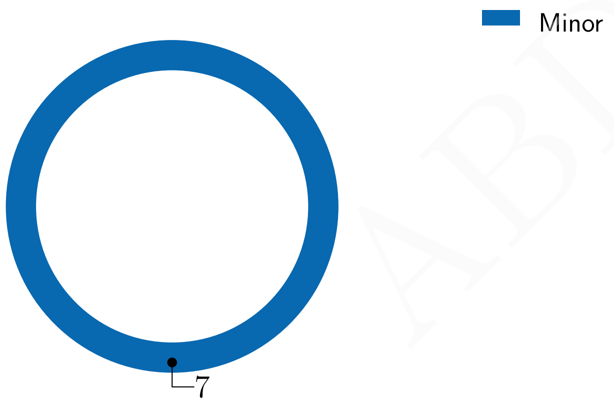
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
29th June 2022

We've been asked to review updates to 5 files in a Github repo. We found 7 minor issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Suboptimal	Info
CVF-2	Minor	Suboptimal	Info
CVF-3	Minor	Suboptimal	Info
CVF-4	Minor	Overflow/Underflow	Info
CVF-5	Minor	Suboptimal	Info
CVF-6	Minor	Suboptimal	Info
CVF-7	Minor	Readability	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 4, 2022	D. Khovratovich	Initial Draft
0.2	June 4, 2022	D. Khovratovich	Minor revision
1.0	June 6, 2022	D. Khovratovich	Release
1.1	June 29, 2022	D. Khovratovich	New fix link and conclusion are added
2.0	June 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've been asked to review the V9 differences in the following files:

- AdditionalZkSync.sol
- Governance.sol
- Storage.sol
- TokenGovernance.sol
- ZkSync.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
- **Category** Suboptimal
- **Status** Info
- **Source** Governance.sol

Description This check is redundant, as it is anyway possible to set a dead governor address.
Client Comment Setters of address type parameters should include a zero-address check otherwise contract functionality may become inaccessible. All in all, it is safer to have this check for accidentally used incorrect zero-address.

Listing 1:

```
+require(_newGovernor != address(0), "1n");
```

3.2 CVF-2

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

Description This change is actually not required, as even without this change, it was possible to query the mapping value off-chain from the contract's storage.

Listing 2:

```
-mapping(address => mapping(uint32 => uint256)) internal
  ↪ authFactsResetTimer;
+mapping(address => mapping(uint32 => uint256)) public
  ↪ authFactsResetTimer;
```

3.3 CVF-3

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

Description These checks don't directly map to any of the vulnerabilities being fixed. They seems redundant, as it is anyway possible to pass an invalid token address.
Recommendation Consider either removing them or explaining their role.
Client Comment It is related to another vulnerability in which it was possible to add a zero-address token with nonzero token id, while zero-address should be reserved for ETH with zero id.

Listing 3:

```
72 +require(_token != address(0), "z1"); // Token should have a non
   ↪ -zero address
   +require(_token != $(ZKSYNC_ADDRESS), "z2"); // Address of the
   ↪ token cannot be the same as the address of the main zksync
   ↪  contract
```

3.4 CVF-4

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

Description Overflow is possible when converting to "uint16".
Recommendation Consider using safe conversion.
Client Comment It is ture that only becasue of the particular value of a configuration constant, a fungible token ID always fits into 16 bits, but 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 4:

```
571 +increasePendingBalance(uint16(op.tokenId), op.owner, op.amount)
    ↪ ;

577 +increasePendingBalance(uint16(op.tokenId), op.target, op.amount
    ↪ );

582 +     increasePendingBalance(uint16(op.tokenId), op.owner, op.
    ↪ amount);
```

3.5 CVF-5

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

Description This linear search is inefficient.
Recommendation Consider allowing the caller to pass a hint with the index inside the "_committedBlocks" array of the first unverified block.
Client Comment It can be changed but with a big legacy or more inconsistency.

Listing 5:

```
+while (hashStoredBlockInfo(_committedBlocks[i]) !=
    ↪ firstUnverifiedBlockHash) {
```

3.6 CVF-6

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

Recommendation Instead of breaking the "for" loop into two "while" loops, it would be simpler to just silently skip committed blocks that don't pass this check, rather than revert on them.
Client Comment In that case, it will be possible to bypass the "for" loop maliciously, and directly jumps to "verfier" code. Then, it is necessary to add another boolean check whether any block passed that check or not. All in all, it will have almost the same complexity.

Listing 6:

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

3.7 CVF-7

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

Description In some cases increment is now done as "x++" while in the other as "x += 1;".
Recommendation Consider using a consistent approach.
Client Comment That is correct. But, we may ignore it for the moment.

Listing 7:

```
830  -            priorityOperationsProcessed++;
     +            ++priorityOperationsProcessed;

862  -                  priorityOperationsProcessed++;
     +                  ++priorityOperationsProcessed;

1082 -totalOpenPriorityRequests++;
     +totalOpenPriorityRequests += 1;
```