Report

v. 1.0

Customer

Term Structure

Smart Contract Audit

Term Structure. Part 2

27th September 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	27.09.23	A. Zveryanskaya	Initial Draft
0.2	27.09.23	A. Zveryanskaya	Minor revision
1.0	27.09.23	A. Zveryanskaya	Release

2 Introduction

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

Term Structure is a decentralized bond protocol powered by zkTrue-up, a platform that enables peer-to-peer lending and borrowing with fixed interest rates.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

```
/
	AccountFacet.sol	EvacuVerifier.sol	ILoanFacet.sol
	LoanFacet.sol	RollupFacet.sol	Utils.sol
	Verifier.sol
```

4 Methodology

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

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

We classify issues by the following severity levels:

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

5 Our findings

We found 1 critical, 4 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Issues	Info	Fixed
Severity: Critical	0	1
Major	0	4
Moderate	1	7
Minor	7	12

Fixed 24 out of 32 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** RollupFacet.sol

Description In case there are less than 32 bytes left between ”curr” and ”end”, this would read garbage after ”pubData” and then revert in case this garbage is not zero.

Client Comment *Fixed to check calldata which only including pubData.*

```
643 let data := mload(curr)
```

7 Major Issues

CVF-3. FIXED

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

Description It is unclear whether it is possible to obtain the account ID for an account address, but if it is, then the “accountId” argument is redundant.

Client Comment *The accountId parameter is for the account which is de-registered in consumeL1Request function, the de-register process remove the address ⇒ accountId mapping but leave accountId ⇒ address maaping for de-registered address to refund. so the de-registered address should use accountId as input params to withdraw.*
We have move this edge case to a new function refundDeregisteredAddr in RollupFacet.sol and remove the accountId from withdraw function parameters to keep general withdraw function simply.

```
88 address accountAddr = asl.getAccountAddr(accountId);
   if (accountAddr != msg.sender) revert AccountAddrIsNotSender(
      ↪ accountAddr, msg.sender);
```

CVF-4. FIXED

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

Description It is unclear, whether the ”isLoanOwner” function returns a boolean value or reverts in case the given address is not the owner of the given loan.

Recommendation Consider double checking.

```
161 msg.sender.isLoanOwner(loanInfo);
```

CVF-7. FIXED

- **Category** Overflow/Underflow
- **Source** Verifier.sol

Description Underflow is possible during subtraction. Should be: mod(sub(q, mod(calldataload(add(pA,32)), q)), q) if out-of-range values should be supported; otherwise the code should revert on such inputs

```
mstore(add(_pPairing, 32), mod(sub(q, calldataload(add(pA, 32))), q)
    ↪ )
```

CVF-8. FIXED

- **Category** Overflow/Underflow
- **Source** EvacuVerifier.sol

Description Underflow is possible during subtraction. Should be: mod(sub(q, mod(calldataload(add(pA,32)), q)), q) if out-of-range values should be supported; otherwise the code should revert on such inputs.

```
mstore(add(_pPairing, 32), mod(sub(q, calldataload(add(pA, 32))), q)
    ↪ )
```

8 Moderate Issues

CVF-2. FIXED

- **Category** Suboptimal
- **Source** RollupFacet.sol

Description A bit more efficient way to ensure all array elements have the same value would be to calculate bitwise AND and bitwise OR of the elements and then ensure they are the same. Another efficient way would be to split array into chunks of some standard sized (say powers of 2), calculate hashes of the chunks and them comparing these hashes to the known expected hashes.

612
```
for (uint256 i = 1; i < chunkIdDeltaLength; ++i) {
    if (chunkIdDeltas[i] != Config.EVACUATION_CHUNK_SIZE) revert
       ↪ InvalidChunkIdDelta(chunkIdDeltas);
}
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** RollupFacet.sol

Description This should be calculated only if "isExecutedL1RequestNumEqTotalL1Request-Num" is false.

669
```
bool isLastL1RequestEvacuation = rsl.getL1Request(lastL1RequestId).
   ↪ opType == Operations.OpType.EVACUATION;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** RollupFacet.sol

Description Using the free memory pointer is redundant here, as the scratch space would be enough and also because the function is going to revert anyway. Just use zero memory pointer.

647
```
let ptr := mload(0x40)
```

CVF-10. FIXED

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

Description IT is unclear, why the case when the last L1 request is EVACUATION is so special, while the case when EVACUATION request is not the last one but still exists among non-executed requests is not so special. In case this is fine, consider adding more details to the comment.

Client Comment *Added clear comments to explain.*

195
```
///  the last L1 request cannot be evacuation because it would means
   ↪  all L1 requests have been consumed and start to evacuate
if (rsl.getL1Request(lastL1RequestId).opType == Operations.OpType.
   ↪ EVACUATION)
    revert LastL1RequestIsEvacuation(totalL1RequestNum);
```

CVF-11. FIXED

- **Category** Suboptimal
- **Source** Verifier.sol

Description There is only one public signal, so this line checks some garbage.

Recommendation Consider removing this line.

173
```
checkField(calldataload(add(_pubSignals, 32)))
```

CVF-12. FIXED

- **Category** Suboptimal
- **Source** EvacuVerifier.sol

Description There is only one public signal, so this line checks some garbage.

Recommendation Consider removing this line.

173
```
checkField(calldataload(add(_pubSignals, 32)))
```

CVF-13. FIXED

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

Description This condition may also be true in case all non-executed L1 requests were already consumed, but the ”evacuate” function wasn’t called yet. This conditino also doesn’t guarantee that all evacuation requests were already committed.

Client Comment *Add comments to explain it should including the scenario that evacuation request is empty.*

```
326 /// If executed L1 requests number == total L1 requests number
    /// means all evacuation requests have been executed,
    /// the protocol will exit the evacuation mode and back to normal
        ↪ mode
    if (rsl.getExecutedL1RequestNum() == rsl.getTotalL1RequestNum()) {
```

CVF-14. INFO

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

Description This flag doesn’t seems to be reset after finishing evacuation and returning to normal operation, so in case of another evacuation, it will be already set.

Recommendation Consider fixing this or explaining in a comment why this is fine.

Client Comment *Actually the state will be changed when we execute evacuation block _executeOneBlock(...) { ... else if (opType == Operations.OpType.EVACUATION) { Operations.Evacuation memory evacuation = pubData.readEvacuationPubdata(); rsl.evacuated[evacuation.accountId][evacuation.tokenId] = false; } }*

```
357 return RollupStorage.layout().isEvacuated(accountId, tokenId);
```

9 Minor Issues

CVF-15. FIXED

- **Category** Documentation
- **Source** Verifier.sol

Description This comment is confusing. The function actually does: pR += (x, y) * s”

```
84 // G1 function to multiply a G1 value(x,y) to value in an address
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** Verifier.sol

Description Adding zero to “pubSignals” seems like waste of gas.

Recommendation Consider removing this operation.

```
119 g1_mulAccC(_pVk, IC1x, IC1y, calldataload(add(pubSignals, 0)))
```

CVF-17. FIXED

- **Category** Suboptimal
- **Source** EvacuVerifier.sol

Description Adding zero to “pubSignals” seems like waste of gas.

Recommendation Consider removing this operation.

```
119 g1_mulAccC(_pVk, IC1x, IC1y, calldataload(add(pubSignals, 0)))
```

CVF-18. FIXED

- **Category** Documentation
- **Source** Verifier.sol

Recommendation Consider adding a comment describing this operation: _pVk += {IC1x, IC1y} * pubSignals [0]

```
119 g1_mulAccC(_pVk, IC1x, IC1y, calldataload(add(pubSignals, 0)))
```

CVF-19. FIXED

- **Category** Documentation
- **Source** EvacuVerifier.sol

Recommendation Consider adding a comment describing this operation: _pVk += {IC1x, IC1y} * pubSignals [0]

```
119 g1_mulAccC(_pVk, IC1x, IC1y, calldataload(add(pubSignals, 0)))
```

CVF-20. FIXED

- **Category** Documentation
- **Source** Verifier.sol

Recommendation Consider adding a comment describing this operation: _pVk = {IC0x, IX0y}

```
114 mstore(_pVk, IC0x)
    mstore(add(_pVk, 32), IC0y)
```

CVF-21. FIXED

- **Category** Documentation
- **Source** EvacuVerifier.sol

Recommendation Consider adding a comment describing this operation: _pVk = {IC0x, IX0y}

```
114 mstore(_pVk, IC0x)
    mstore(add(_pVk, 32), IC0y)
```

CVF-22. INFO

- **Category** Suboptimal
- **Source** Utils.sol

Description It would be more efficient to have two versions of each function: one for TSB tokens and another for normal tokens.

Client Comment *If split into two functions, we still need to select a function by token type when transferring tokens, so merging into one function looks better generally.*

53
```
/// @param isTsbToken The flag to indicate whether the token is Tsb
    ↪ Token
```

68
```
/// @param isTsbToken The flag to indicate whether the token is Tsb
    ↪ Token
```

CVF-23. FIXED

- **Category** Documentation
- **Source** RollupFacet.sol

Description There is only one assertion below.

Client Comment *Fixed the comments, it means that we combine this two assertion to one assertion in the code.*

551
```
/// Two assertions below are equivalent to:
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** Verifier.sol

Description This assignment is redundant, as memory is used only temporary and the function always terminates the transaction.

Client Comment *Keep to follow snarkJS template originally.*

166
```
let pMem := mload(0x40)
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** EvacuVerifier.sol

Description This assignment is redundant, as memory is used only temporary and the function always terminates the transaction.

Client Comment *Keep to follow snarkJS template originally.*

```
167 mstore(0x40, add(pMem, pLastMem))
```

CVF-26. FIXED

- **Category** Suboptimal
- **Source** RollupFacet.sol

Description This assumes that public data length is in bits rather than in bytes, which seems very unusual. If this is true, consider clearly stating this in a comment.

Client Comment *The assertion is derived from two assertion and the function and comments are moved to RollupLib.requireValidPubDataLength.*

```
556 uint256 publicDataLength = newBlock.publicData.length;
    if (publicDataLength % Config.BITS_OF_CHUNK != 0) revert
       ↪ InvalidPubDataLength(publicDataLength);
```

CVF-27. FIXED

- **Category** Suboptimal
- **Source** RollupFacet.sol

Description This branch is redundant.

Recommendation Consider removing it.

```
227 } else {
        // do nothing, others L1 requests have no storage changes
    }
```

CVF-28. FIXED

- **Category** Documentation
- **Source** EvacuVerifier.sol

Description This comment is confusing. The function actually does: pR += (x, y) * s

```
84 // G1 function to multiply a G1 value(x,y) to value in an address
```

CVF-29. INFO

- **Category** Suboptimal
- **Source** LoanFacet.sol

Description This event is emitted even if nothing actually changed.

Client Comment *They are admin function and no critical effect although no change.*

```
215 emit SetActivatedRoller(isActivated);
```

CVF-31. FIXED

- **Category** Suboptimal
- **Source** LoanFacet.sol

Description This variable is redundant as it is used only once.

```
157 bool isActivated = lsl.getRollerState();
```

CVF-32. INFO

- **Category** Procedural
- **Source** RollupFacet.sol

Description We didn't review this function.

Client Comment *Minor update to use processRequestFunc as input params.*

```
293 _commitBlocks(rsl, lastCommittedBlock, evacuBlocks, true);
```

CVF-33. INFO

- **Category** Procedural
- **Source** RollupFacet.sol

Description We didn't review this function.

```
308 _verifyBlocks(rsl, evacuBlocks);
```

CVF-34. INFO

- **Category** Procedural
- **Source** RollupFacet.sol

Description We didn't review this function.

```
324 _executeBlocks(rsl, evacuBlocks);
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting