Report

v. 1.0

Customer

Lombard Finance

Smart Contract Audit

Stake and Bake

22nd September 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	22.09.25	A. Zveryanskaya	Initial Draft
0.2	22.09.25	A. Zveryanskaya	Minor revision
1.0	22.09.25	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

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.

The Lombard team was assembled from leaders at Polychain, Coinbase, Ripple, Maple, Argent and bring deep expertise across crypto, finance, and engineering — united by the belief that a permissionless Bitcoin economy isn't just possible, it's necessary.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/	
StakeAndBake.sol	
depositor/	
IDepositor.sol	TellerWithMultiAssetSupportDepositor.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 whether the code actually does what it is supposed to do, whether the algorithms are optimal and correct, and whether 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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 3 major, and a few less important issues. All identified Major issues have been fixed.

Fixed 3 out of 3 issues

6 Major Issues

CVF-1 FIXED

- **Category** Documentation
- **Source** StakeAndBake.sol

Description Despite the comment, this call doesn't seem to send anything to the owner.

Recommendation Fix the comment.

```
157 // First, mint the LBTC and send to owner.
    $.lbtc.mint(data.mintPayload, data.proof);
```

CVF-2 FIXED

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

Description The returned value is ignored.

Recommendation Explicitly check that true was returned or use the "safeTransferFrom" function.

```
190 $.lbtc.transferFrom(owner, address(this), permitAmount);
```

CVF-3 FIXED

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

Description The returned value is ignored.

Recommendation Explicitly check that true was returned or use the "safeTransfer" function.

```
194 if (feeAmount > 0) $.lbtc.transfer($.lbtc.getTreasury(), feeAmount);
```

7 Moderate Issues

CVF-4 FIXED

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

Description There is no range check for this argument.

Recommendation Add proper range check.

```
84 uint256 fee_,
```

CVF-5 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Recommendation There should be “depositorSet” modifier here.

```
136 ) external onlyRole(CLAIMER_ROLE) whenNotPaused {
```

CVF-6 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Description A failed call here may consume all remaining gas, effectively making it impossible to report error and proceed to other calls.

Recommendation Limit the gas amount per call.

```
138 try this.stakeAndBake(data[i]) {} catch {
```

CVF-7 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Recommendation Including the call data into an error doesn’t make much sense, as the caller anyway knows what data he passed. More helpful it would be to include the error message from the failed call.

```
139 emit BatchStakeAndBakeReverted(i, data[i]);
```

CVF-8 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Description Calling the “stakeAndBake” function here is inefficient, as this function performs checks already performed, such as claimer role check and not paused check.

Recommendation Call instead a special version of the “stakeAndBake” function with only one check: msg.sender == this

```
138 try this.stakeAndBake(data[i]) {} catch {
```

CVF-9 INFO

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Description This code seems to be aware to the “mintPayload“ internal structure.

Recommendation Decompose the “mintPayload” field into several fields to make its internal structure transparent.

Client Comment *We are utilizing the same approach in many other contracts and do not consider this as a problem. Prefer to keep it the way it is right now*

```
173 data.mintPayload[4:]
```

CVF-10 FIXED

- **Category** Procedural
- **Source** StakeAndBake.sol

Description Here an implicit underflow check is used to enforce a business-level constraint. This is a bad practice, as it makes code harder to read and more error prone. Also, implicit checks are less efficient, as they consume all gas when fail.

Recommendation Add and explicit check to ensure permit amount is enough to at least pay fee.

```
196 uint256 remainingAmount = permitAmount - feeAmount;
```

CVF-11 FIXED

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

Description The returned value is ignored.

Recommendation Explicitly check that true was returned or use the “safeApprove” function.

```
201 $.lbtc.approve(address($.depositor), remainingAmount);
```

8 Recommendations

CVF-12 INFO

- **Category** Procedural
- **Source** IDepositor.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Specify as "^0.,8.0" or as "^0.8.24" if there is something special regarding this particular version. Also relevant for: TellerWithMultiAssetSupportDepositor.sol, StakeAndBake.sol.

Client Comment *We prefer to set explicit Solidity version to make cmpile results more predictable and working across many different EVM-compatible chains*

```
2 pragma solidity 0.8.24;
```

CVF-13 FIXED

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

Recommendation The function should also return as a bytes array the result of the vault deposit call.

```
16  * @param depositPayload Optional ABI encoded parameters needed for
      ↪ a vault deposit call
18 function deposit(
```

CVF-14 INFO

- **Category** Readability
- **Source** TellerWithMultiAssetSupportDepositor.sol

Recommendation Consider putting all OpenZeppelin imports together.

Client Comment *The recommendation is not clear.*

```
4 import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
  import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/
    ↪ SafeERC20.sol";

7 import {ReentrancyGuard} from "@openzeppelin/contracts/utils/
    ↪ ReentrancyGuard.sol";
```

CVF-15 FIXED

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

Recommendation The type for this variable should be “ITeller”.

```
23 address public immutable teller;
```

CVF-16 FIXED

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

Recommendation The type for this variable should be “IERC20”.

```
24 address public immutable depositAsset;
```

CVF-17 FIXED

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

Recommendation The type for the “teller_” argument should be “ITeller”.

```
27 constructor(address teller_, address depositAsset_, address
    ↪ stakeAndBake_) {
```

CVF-18 FIXED

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

Recommendation The type for the “depositAsset_” argument should be “IERC20”.

```
27 constructor(address teller_, address depositAsset_, address
   ↪ stakeAndBake_) {
```

CVF-19 FIXED

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

Description This function should somehow return the actual number of shares issued.

```
47 function deposit(
```

CVF-20 FIXED

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

Description This call is expensive. Is it really necessary to query destination on each deposit?

```
62 address vault = destination();
```

CVF-21 FIXED

- **Category** Procedural
- **Source** TellerWithMultiAssetSupportDepositor.sol

Recommendation This interface should be moved into a separate file named “ITeller.sol”.

```
89 interface ITeller {
```

CVF-22 FIXED

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

Recommendation The type for this argument should be “IERC20”.

```
91 ERC20 depositAsset,
```

CVF-23 FIXED

- **Category** Bad naming
- **Source** TellerWithMultiAssetSupportDepositor.sol

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value and/or explain in a documentation comment.

```
94 ) external returns (uint256);
```

CVF-24 FIXED

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

Recommendation The return type for this function should be “IERC20” or other specific type.

```
96 function vault() external view returns (address);
```

CVF-25 INFO

- **Category** Procedural
- **Source** StakeAndBake.sol

Description We didn’t review these files.

```
4  import {LBTC} from "../LBTC/LBTC.sol";
   import {ILBTC} from "../LBTC/ILBTC.sol";
```

```
10 import {Actions} from "../libs/Actions.sol";
```

CVF-26 INFO

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Recommendation This error doesn't have much sense, as it is anyway possible to change operator to a dead address.

Client Comment *It might be helpful in case function gets mistakenly called with parameters not initalized or set.*

```
25 /// @dev error thrown when operator is changed to zero address
   error ZeroAddress();
```

CVF-27 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Recommendation This error could be made more useful by adding the invalid fee value as a parameter.

```
28 error FeeGreaterThanMaximum();
```

CVF-28 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

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

```
37 event FeeChanged(uint256 indexed oldFee, uint256 indexed newFee);
```

CVF-29 FIXED

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Description Indexing numeric parameters doesn't make much sense, as indexes only helps with exact value lookups.

Recommendation Don't index parameters in this event.

```
37 event FeeChanged(uint256 indexed oldFee, uint256 indexed newFee);
```

CVF-30 FIXED

- **Category** Documentation
- **Source** StakeAndBake.sol

Description Repeating field names in the comments is redundant.

Recommendation Remove field names from the comments.

```
40 /// @notice permitPayload Contents of permit approval signed by the
      ↪ user
   bytes permitPayload;
   /// @notice depositPayload Contains the parameters needed to
      ↪ complete a deposit
   bytes depositPayload;
   /// @notice mintPayload The message with the stake data
   bytes mintPayload;
   /// @notice proof Signature of the consortium approving the mint
   bytes proof;
```

CVF-31 INFO

- **Category** Suboptimal
- **Source** StakeAndBake.sol

Description Solidity compiler is smart enough to precompute constant hash expression.

Recommendation Replace hardcoded hash with an expression.

Client Comment *We utilize the same approach in almost all our contracts and consider it helpful in case we need to check the storage manually. Anyway there is no any difference in terms of how the contract works.*

```
61 // keccak256(abi.encode(uint256(keccak256("lombardfinance.storage.
      ↪ StakeAndBake")) - 1)) & ~bytes32(uint256(0xff))
   bytes32 private constant STAKE_AND_BAKE_STORAGE_LOCATION =
   0xd0321c9642a0f7a5931cd62db04cb9e2c0d32906ef8824eece128a7ad5e4f500;
```

CVF-32 FIXED

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

Recommendation The type for this argument should be “LBTC”.

```
81 address lbtc_,
```

CVF-33 INFO

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

Description This event is emitted even if nothing actually changed.

```
115 emit FeeChanged(oldFee, fee);
```

```
128 emit DepositorSet(depositor);
```

CVF-34 INFO

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

Recommendation The type for this argument should be “IDepositor”.

```
123 address depositor
```

CVF-35 FIXED

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

Description This function should return the amount of shares issued for each element of the batch.

```
134 function batchStakeAndBake(
```

CVF-36 FIXED

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

Description This functions should return the amount of shares issued.

```
152 function stakeAndBake(
```

CVF-37 FIXED

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

Recommendation The return type should be “IDepositor”.

```
212 function getStakeAndBakeDepositor() external view returns (address)
    ↪ {
```

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