Report

v. 1.0

Customer

zkLend

Circuit Audit

zkLend Core

18th August 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	17.08.23	A. Zveryanskaya	Initial Draft
0.2	17.08.23	A. Zveryanskaya	Minor revision
1.0	18.08.23	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.

zkLend is a money-market protocol combining zk-rollup scalability, superior transaction speed, and cost-savings with Ethereum's security.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	Market.cairo	PriceOracle.cairo	Proxy.cairo
	ZToken.cairo		
interfaces/callback/			
	IZklendFlash Callback.cairo		
interfaces/third_parties/			
	IEmpiricOracle.cairo		
interfaces/			
	IInterestRateModel.cairo	IMarket.cairo	IPriceOracle.cairo
	IPriceOracleSource.cairo	IZToken.cairo	
internals/Market/			
	events.cairo	functions.cairo	storage.cairo
	structs.cairo		
internals/ZToken/			
	events.cairo	functions.cairo	storage.cairo
irms/			
	DefaultInterest RateModel.cairo		
libraries/			
	Math.cairo	SafeCast.cairo	SafeDecimalMath.cairo
	SafeMath.cairo		
oracles/			
	Empiric OracleAdapter.cairo		

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 19 Moderate, and a few less important issues.

	Info	Fixed
Moderate	19	1
Minor	64	2

Fixed 3 out of 86 issues

6 Moderate Issues

CVF-14. INFO

- **Category** Unclear behavior
- **Source** DefaultInterestRateModel.cairo

Description The interest rate model implemented in this function is stateless.

Recommendation A stateful model could be more reasonable. Seee the following note for details: https://hackmd.io/@abdk/S1c9qilGi

Client Comment *Can't comment. We don't seem to have access to the linked MD file.*

```
65 func calculate_borrow_rate{syscall_ptr: felt*, pedersen_ptr:
   ↪ HashBuiltin*, range_check_ptr}(
```

CVF-20. INFO

- **Category** Procedural
- **Source** functions.cairo

Description ERC-20 standard doesn't suppose the "Approval" event to be emitted by the "transferFrom" function.

Recommendation Consider removing this line.

Client Comment *Disagreed. While it's true that ERC20 itself does not require (https://eips.ethereum.org/EIPS/eip-20#approval) it to be emitted, it's also not forbidden. Additionally, popular implementations like OpenZeppelin (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/d00acef4059807535af0bd0dd0ddf619747a044b/contracts/token/ERC20/ERC20.sol#L158-L167) emits the event on 'transferFrom'.*

```
119 Approval.emit(sender, caller, new_allowance_u256);
```

CVF-41. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description Using annual rates makes code more complicated and makes calculations less precise.

Recommendation Consider using per-second rates instead. Conversion between annual rates and per-second rates could be done in UI to completely hide per-second rates from users.

Client Comment *Noted but won't fix. We prefer not to refactor this part of the code before the mainnet launch as we're afraid we won't have enough time to thoroughly test the changes.*

```
358 let temp_3 = SafeMath.div(temp_2, SECONDS_PER_YEAR);
```

```
382 let temp_2 = SafeMath.div(temp_1, SECONDS_PER_YEAR);
```

```
415 let temp_3 = SafeMath.div(temp_2, SECONDS_PER_YEAR);
```

CVF-42. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description The new debt accumulator value is rounded down here, i.e. towards borrowers. It is a good practice to always round towards the protocol to make it impossible abusing rounding errors.

Recommendation Consider rounding up.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

```
384 let latest_accumulator = SafeDecimalMath.mul(temp_3, reserve.
    ↪ debt_accumulator);
```

CVF-44. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description This function calculates the user debt rounded down, i.e. towards the user. A good practice is to always round towards the protocol.

Recommendation Consider rounding up.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

437
```
func get_user_debt_for_token{syscall_ptr: felt*, pedersen_ptr:
   ↪ HashBuiltin*, range_check_ptr}(
```

CVF-46. INFO

- **Category** Procedural
- **Source** functions.cairo

Recommendation This check should be done earlier, before the first use of "token".

Client Comment *Noted but won't fix until Cairo 1 rewrite, as failed transactions can't possibly be charged fees in Cairo 0, so this won't actually affect users.*

507
```
Internal.assert_reserve_enabled(token);
```

570
```
Internal.assert_reserve_enabled(token);
```

CVF-49. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description Here the scaled amount is rounded down, i.e. toward user. It is a good practice to always round towards the protocol.

Recommendation Consider rounding up.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

572
```
let scaled_down_amount = SafeDecimalMath.div(amount,
   ↪ updated_debt_accumulator);
```

CVF-54. INFO

- **Category** Procedural
- **Source** functions.cairo

Recommendation This check should be done earlier, before repaying the debt.

Client Comment *Noted but won't fix until Cairo 1 rewrite, as failed transactions can't possibly be charged fees in Cairo 0, so this won't actually affect users.*

```
708 let (is_collateral) = is_used_as_collateral(user, collateral_token);
    with_attr error_message("Market: cannot withdraw non-collateral
       ↪ token") {
710     assert is_collateral = TRUE;
    }
```

CVF-62. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here, the required collateral value is rounded down, i.e. towards user. A good practice is to always round towards the protocol.

Recommendation Consider rounding up.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

```
1035 let collateral_required = SafeDecimalMath.div(debt_value,
        ↪ borrow_factor);
```

CVF-63. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here, the scaled debt balance is calculated rounded down, i.e. towards user.

Recommendation Consider rounding up, i.e. towards the protocol.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

```
1053 let scaled_up_debt_balance = SafeDecimalMath.mul(raw_debt_balance,
        ↪ debt_accumulator);
```

CVF-65. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here, the debt value is calculated rounded down, i.e. towards user.

Recommendation Consider rounding up, i.e. towards the protocol.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

1060
```
let debt_value = SafeDecimalMath.mul_decimals(debt_price,
    ↪ scaled_up_debt_balance, decimals);
```

CVF-66. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here, the product of "collateral_price" and "collateral_balance" is divided by 10^reserve.decimals and then immediately multiplied by "reserve.collateral_factor". This could lead to loss of precision.

Recommendation Consider doing all the divisions at the very end.

Client Comment *Noted but won't fix. We prefer not to refactor this part of the code before the mainnet launch as we're afraid we won't have enough time to thoroughly test the changes.*

1079
```
let collateral_value = SafeDecimalMath.mul_decimals(
```

1083
```
let discounted_collateral_value = SafeDecimalMath.mul(
```

CVF-68. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here "repay_amount" is rounded down, i.e. towards user.

Recommendation Consider rounding up, i.e. towards the protocol.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

1159
```
let repay_amount = SafeDecimalMath.mul(user_raw_debt,
    ↪ updated_debt_accumulator);
```

CVF-70. INFO

- **Category** Documentation
- **Source** functions.cairo

Description Burning all tokens when zero amount is passed is an unusual behaviour.

Recommendation Consider documenting it or rather using -1 instead.

Client Comment *Noted but won't fix at the moment. This behavior is completely internal and not exposed to external users in any way. We will address this during the Cairo 1 rewrite instead.*

```
1216 if (amount == 0) {
```

CVF-4. INFO

- **Category** Unclear behavior
- **Source** PriceOracle.cairo

Description There is no check that an oracle exists for the token.

Recommendation Consider adding an explicit check for this.

Client Comment *Won't fix. If 'source' turns out to be zero, the following contract calls would fail anyways.*

```
25 let (source) = sources.read(token);
```

```
34 let (source) = sources.read(token);
```

CVF-39. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description This function should update accumulator before setting a new treasury, to make sure that all the treasury value accumulated when the old treasury was set, was transferred to the old treasury.

Client Comment *Noted but won't fix. This is likely negligible, as the accumulators would be updated everytime someone uses the reserve anyways.*

```
303 func set_treasury{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
    ↪ range_check_ptr}(
```

CVF-55. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here, the product of “debt_token_price” and “amount” is first divided by “10^debt_reserve_decimals” and then immediately multiplied by “10^collateral_reserve.decimals”. Such operation could lead to significant loss of precision.

Recommendation Consider calculating like this:
(debt_token_price * amount * 10^collateral_reserve.decimals) / (10^debt_reserve_decimals * collateral_token_price)
or, even better, just multiply by 10^(collateral_reserve.decimals - debt_reserve_decimals) in case collateral has more decimals than debt; or just divide by 10^(debt_reserve_decimals - collateral_reserve.decimals) in case debt has more decimals than collateral.

Client Comment *Noted but won’t fix. We prefer not to refactor this part of the code before the mainnet launch as we’re afraid we won’t have enough time to thoroughly test the changes.*

720
```
let debt_value_repaid = SafeDecimalMath.mul_decimals(
    debt_token_price, amount, debt_reserve_decimals
);
let equivalent_collateral_amount = SafeDecimalMath.div_decimals(
    debt_value_repaid, collateral_token_price, collateral_reserve.
       ↪ decimals
);
```

CVF-56. INFO

- **Category** Flaw
- **Source** functions.cairo

Description In case the liquidation bonus is higher than the collateral factor, a liquidation is virtuall guaranteed to make the situation for the liquidated account even worse than it was before.

Recommendation Consider forbidding setting collateral factor below liquidation bonus.

Client Comment *Noted but won’t fix at the moment. We’re aware of this issue and will simply make sure to be careful when configuring liquidation settings initially. We will instead address this in a future upgrade.*

727
```
let collateral_amount_after_bonus = SafeDecimalMath.mul(
    equivalent_collateral_amount, one_plus_liquidation_bonus
);
```

CVF-58. FIXED

- **Category** Flaw
- **Source** functions.cairo

Description There is no simple way for a callback to ensure that the passed calldata could be trusted.

Recommendation Consider either passing the caller address to the callback, or invoking the callback on the caller rather than on the receiver.

```
778 IZklendFlashCallback.zklend_flash_callback(
        contract_address=receiver, calldata_len=calldata_len, calldata=
            ↪ calldata
780 );
```

CVF-71. INFO

- **Category** Flaw
- **Source** functions.cairo

Description It seems that “lending_accumulator” may never go down, even when the protocol looses money for whatever reason. This means that it is a possible situation when the protocol doesn’t have enought tokens to repay all the deposits in full. This means that in case of a bank run, only those whoe withdraw first will be paid.

Recommendation Consider forbidding withdrawals in case “lending_accumulator” is higher than is should be.

Client Comment *Noted but won’t fix at the moment. We will initially be launching with a borrowing limit so this wouldn’t cause too much damage in any case. We will address this with an additional check to disallow withdrawals if the loss percentage is higher than a certain threshold.*

```
1372 if (no_need_to_adjust == FALSE) {
```

7 Minor Issues

CVF-1. INFO

- **Category** Procedural
- **Source** Proxy.cairo

Description This file is very similar to the following file from OpenZeppelin: https://github.com/OpenZeppelin/cairo-ontracts/blob/main/src/openzeppelin/upgrades/presets/Proxy.cairo

Recommendation Consider using the OpenZeppelin's version or explaining, why a different implementation was necessary.

Client Comment *OpenZeppelin's version does not allow deploying the proxy and calling the initializer in one call. We changed the constructor for that.*

```
3 %lang starknet
```

CVF-2. INFO

- **Category** Procedural
- **Source** Proxy.cairo

Description We didn't review this file.

Client Comment *Noted.*

```
7 from openzeppelin.upgrades.library import Proxy
```

CVF-3. INFO

- **Category** Procedural
- **Source** PriceOracle.cairo

Description We didn't review this file.

Client Comment *Noted.*

```
9 from openzeppelin.access.ownable.library import Ownable
```

CVF-5. FIXED

- **Category** Unclear behavior
- **Source** PriceOracle.cairo

Description This function should emit some event.

```
40 func set_token_source{syscall_ptr: felt*, pedersen_ptr: HashBuiltin
   ↪ *, range_check_ptr}(
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** EmpiricOracleAdapter.cairo

Description This function is just an alias for the “get_data” function.

Recommendation Consider leaving only one of these two functions.

Client Comment *Won’t fix. We prefer separating external and internal functions. ‘get_price_with_time‘ is an external function and ‘get_data‘ is internal.*

```
39 func get_price_with_time{syscall_ptr: felt*, pedersen_ptr:
   ↪ HashBuiltin*, range_check_ptr}() -> (
```

CVF-7. INFO

- **Category** Procedural
- **Source** SafeDecimalMath.cairo

Recommendation These constants must be named.

Client Comment *Noted but won’t fix at the moment. Will address during the Cairo 1 rewrite.*

```
10 const SCALE = 10 ** 27;
```

```
27             assert_le_felt(b_decimals, 75);
```

```
37             assert_le_felt(b_decimals, 75);
```

CVF-8. INFO

- **Category** Overflow/Underflow
- **Source** SafeDecimalMath.cairo

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Recommendation Consider calculating “scaled_product” as a 512-bit value.

Client Comment *Disagreed. It won’t overflow because we’ve limited the number of decimals to be no more than 75.*

```
14 let scaled_product = SafeMath.mul(a, b);
   return SafeMath.div(scaled_product, SCALE);
```

```
30 let scaled_product = SafeMath.mul(a, b);
```

```
32 return SafeMath.div(scaled_product, scale);
```

CVF-9. INFO

- **Category** Overflow/Underflow
- **Source** SafeDecimalMath.cairo

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Recommendation Consider calculating “scaled_a” as a 512-bit value.

Client Comment *Disagreed. It won’t overflow because we’ve limited the number of decimals to be no more than 75.*

```
19 let scaled_a = SafeMath.mul(a, SCALE);
20 return SafeMath.div(scaled_a, b);
```

```
41 let scaled_a = SafeMath.mul(a, scale);
   return SafeMath.div(scaled_a, b);
```

CVF-10. INFO

- **Category** Documentation
- **Source** SafeCast.cairo

Description This code is valid for the particular prime field. Consider mentioning this in the comment

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite if it's still relevant.*

```
30 assert value.low = 0;
```

CVF-11. INFO

- **Category** Documentation
- **Source** Math.cairo

Description This function returns only the lowest 251 bits of the shifted value.

Recommendation Consider mentioning this fact in a comment.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
15 func shl{range_check_ptr, bitwise_ptr: BitwiseBuiltin*}(a: felt, b:
   ↪ felt) -> felt {
```

CVF-12. INFO

- **Category** Unclear behavior
- **Source** DefaultInterestRateModel.cairo

Description There are no range checks for the arguments.

Recommendation Consider adding appropriate checks.

Client Comment *Noted but won't fix at the moment. For now we will simply make sure to be careful when setting interest rate models since these are permissioned anyways.*

```
25 slope_0: felt, slope_1: felt, y_intercept: felt, optimal_rate: felt
```

CVF-13. INFO

- **Category** Documentation
- **Source** DefaultInterestRateModel.cairo

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
54 utilization_rate: felt
```

CVF-15. INFO

- **Category** Unclear behavior
- **Source** DefaultInterestRateModel.cairo

Description Here a division result is multiplied by "params.slope_0" which could lead to precision loss.

Recommendation Consider doing all divisions at the very end.

Client Comment *Disagreed. We decided not to do that because it would overflow even with reasonably-ranged values.*

```
74 let temp_1 = SafeDecimalMath.div(utilization_rate, params.
      ↪ optimal_rate);
   let temp_2 = SafeDecimalMath.mul(params.slope_0, temp_1);
```

```
84 let temp_1 = SafeDecimalMath.div(excess_utilization_rate,
      ↪ optimal_to_one);
   let temp_2 = SafeDecimalMath.mul(params.slope_1, temp_1);
```

CVF-16. INFO

- **Category** Procedural
- **Source** functions.cairo

Description We didn't review these files.

Client Comment *Noted.*

```
28 from openzeppelin.upgrades.library import Proxy
   from openzeppelin.token.erc20.library import Approval, Transfer
```

CVF-17. INFO

- **Category** Documentation
- **Source** functions.cairo

Description It is unclear how name and symbol are packed into field elements.

Recommendation Consider documenting.

Client Comment *Won't fix. We're using native Cairo short strings (https://www.cairo-ang.org/docs/how_cairo_works/consts.html#short-string-literals) here with no custom encoding. It's inline with how popular implementations (e.g. OpenZeppelin) do it.*

```
36 _name: felt,
   _symbol: felt,
```

CVF-18. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description There is not range check for this argument.

Recommendation Consider adding an appropriate check.

Client Comment *Noted but won't fix at the moment. This would be irrelevant after the Cairo 1 rewrite since we would use 'u8'. For now we would simply be careful to use correct values.*

```
38 _decimals: felt,
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description The amounts converted here from uint256 to felt will later be converted back to be used as event parameters.

Recommendation Consider refactoring to avoid reverse conversion.

Client Comment *Noted but won't fix at the moment.*

```
74  let felt_amount = SafeCast.uint256_to_felt(amount);
```

```
101 let felt_amount = SafeCast.uint256_to_felt(amount);
```

```
135 let felt_amount = SafeCast.uint256_to_felt(amount);
```

CVF-21. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Due to rounding errors, the actual increase of the recipient balance could be less than "amount". So this event could be inaccurate. Probably not an issue.

Client Comment *We're aware of this issue. We believe this is an unfortunate side-effect of ERC20 implementations with self-increasing balances.*

```
205 Transfer.emit(0, to, amount_u256);
```

CVF-22. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Due to rounding errors, the actual decrease of the recipient balance could be less than "amount". So this event could be inaccurate. Probably not an issue.

Client Comment *We're aware of this issue. We believe this is an unfortunate side-effect of ERC20 implementations with self-increasing balances.*

```
237 Transfer.emit(user, 0, amount_u256);
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description The “else” branch is redundant.

Recommendation Just do “return ()” after the conditional statement.

Client Comment *Won’t fix. This ”unnecessary” branching was specifically done to work around a lifetime issue. Removing the ‘else‘ branch and returning afterwards causes a compilation error.*

```
456     return ();
} else {
    return ();
}
```

CVF-24. INFO

- **Category** Bad naming
- **Source** storage.cairo

Recommendation Consider naming all offset constants based on the semantics of corresponding structure fields, so that it becomes easier to modify the data structure.

Client Comment *Noted but won’t fix at the moment. Cairo 1 has vastly different storage code, so we believe fixing it here wouldn’t be very meaningful. We will take this into account when implementing something similar in the Cairo 1 rewrite.*

```
15 namespace reserves {
```

CVF-25. INFO

- **Category** Unclear behavior
- **Source** storage.cairo

Description It is unclear why one would need to cast the address of a value to a felt array just to get the first element of this array, which would be just the original value.

Client Comment *Won’t fix. This is automatically generated code by the compiler. We simply extracted those into a module so that we can add methods other than ‘addr‘, ‘read‘, and ‘write‘. Keeping the code as-is from compiler output makes it easier to validate.*

```
23 let (res) = hash2{hash_ptr=pedersen_ptr}(res, cast(&token, felt*)
   ↪ [0]);
```

CVF-26. INFO

- **Category** Suboptimal
- **Source** storage.cairo

Description This code relies on the fact that tempvars above allocated sequentially without gaps, which may not be the case. It would be better not to rely on such things.

Client Comment *Won't fix. This is automatically generated code by the compiler. We simply extracted those into a module so that we can add methods other than 'addr', 'read', and 'write'. Keeping the code as-is from compiler output makes it easier to validate.*

```
69 return ([cast(&__storage_var_temp0, Structs.ReserveData*)],);
```

CVF-27. INFO

- **Category** Documentation
- **Source** storage.cairo

Description It is unclear how name and symbol are packed into field elements.

Recommendation Consider documenting.

Client Comment *Won't fix. We're using native Cairo short strings (https://www.cairo-ang.org/docs/how_cairo_works/consts.html#short-string-literals) here with no custom encoding. It's inline with how popular implementations (e.g. OpenZeppelin) do it.*

```
14 func token_name() -> (name: felt) {
```

```
18 func token_symbol() -> (symbol: felt) {
```

CVF-28. INFO

- **Category** Documentation
- **Source** storage.cairo

Description It is unclear what "raw" means here.

Recommendation Consider explaining in a comment.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
26 func raw_total_supply() -> (total_supply: felt) {
```

```
30 func raw_balances(account: felt) -> (balance: felt) {
```

CVF-29. INFO

- **Category** Documentation
- **Source** events.cairo

Description The semantics of this event and its parameters is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
6 func RawTransfer(from_: felt, to: felt, raw_value: felt, accumulator
    ↪ : felt, face_value: felt) {
```

CVF-30. INFO

- **Category** Documentation
- **Source** structs.cairo

Description It is unclear what fields this property applies to.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
6 decimals: felt,
```

CVF-31. INFO

- **Category** Unclear behavior
- **Source** structs.cairo

Description The "decimals" property makes the whole model more complicated, while the felt range is large enough to use the same precision for all values.

Recommendation Consider using the same precision, e.g. 18 decimals or 64 bits, and remove the "decimals" property.

Client Comment *Noted but won't fix. Unfortunately we have to deal with decimals at some point since we need to deal with prices from oracles, which tend to be decimal-aware. Therefore, we need to handle decimals somewhere, and we believe it's easier to do it here.*

```
6 decimals: felt,
```

CVF-32. INFO

- **Category** Documentation
- **Source** structs.cairo

Description The number format of these values is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
collateral_factor: felt,
borrow_factor: felt,
reserve_factor: felt,
```

```
current_lending_rate: felt,
current_borrowing_rate: felt,
```

CVF-33. INFO

- **Category** Documentation
- **Source** structs.cairo

Description It is unclear, whether these values are absolute or percentages. 'in the latter case, the number format of these values is unclear.

Recommendation Consider documenting or making the names more descriptive.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
flash_loan_fee: felt,
liquidation_bonus: felt,
```

CVF-34. INFO

- **Category** Procedural
- **Source** functions.cairo

Description We didn't review these files.

Client Comment *Noted.*

```
52 from openzeppelin.access.ownable.library import Ownable
   from openzeppelin.security.reentrancyguard.library import
      ↪ ReentrancyGuard
   from openzeppelin.token.erc20.IERC20 import IERC20
   from openzeppelin.upgrades.library import Proxy, Proxy_initialized
```

CVF-35. INFO

- **Category** Procedural
- **Source** functions.cairo

Description This number is confusing.

Recommendation Consider explaining how it was derived and/or giving the hexadecimal equivalent in a comment.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
59 const DEBT_FLAG_FILTER =
      ↪ 1206167596222043702328864427173832373471562340267089208744349833415761767 0
      ↪
```

CVF-36. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description This logic duplicaates the initialization logic of “Proxy”, but uses owner instead of proxy admin.

Recommendation Consider just calling the “Proxy.initializer” function passing “owner” as an argument.

Client Comment *Disagreed. We want to use ‘owner‘ for both contract operation (e.g. adding reserves) and upgrades, and calling ‘Proxy.initializer()‘ here would result in duplicate storage slots.*

```
65 let (initialized) = Proxy_initialized.read();
   with_attr error_message("Proxy: contract already initialized") {
       assert initialized = FALSE;
   }
   Proxy_initialized.write(TRUE);
```

```
76 Ownable.initializer(owner);
```

CVF-37. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Recommendation This check should be performed earlier, right after calculating the new reserve count.

Client Comment *Noted but won’t fix at the moment. It’s extremely unlikely we will be affected by this issue.*

```
296 with_attr error_message("Market: too many reserves") {
        assert_le_felt(new_reserve_count, 125);
    }
```

CVF-38. INFO

- **Category** Bad datatype
- **Source** functions.cairo

Recommendation The value “125” should be a named constant.

Client Comment *Noted but won’t fix at the moment. Will address during the Cairo 1 rewrite.*

```
297 assert_le_felt(new_reserve_count, 125);
```

CVF-40. INFO

- **Category** Procedural
- **Source** functions.cairo

Recommendation This assignment should be moved into the “else” branch of the conditional statement above. For the “then” branch, just assign “SCALE”.

Client Comment *Noted but won’t fix. Doing that somehow triggers a mysterious compiler bug that fails the compilation. This should be irrelevant for Cairo 1 though.*

```
354 let one_minus_reserve_factor = SafeMath.sub(SCALE,
    ↪ effective_reserve_factor);
```

CVF-43. INFO

- **Category** Procedural
- **Source** functions.cairo

Description Unlike other similar functions, these functions return scaled debt amounts.

Recommendation Consider reflecting this fact in the function names or clearly explaining in comments.

Client Comment *Noted but won’t fix at the moment. While we believe more documentation is always better, here we’re consistent that all external functions only return/accept scaled amounts, unless explicitly specified. This is because we think the whole ”scaling” is an implementation detail that users shouldn’t need to care about.*

```
424 func get_total_debt_for_token{syscall_ptr: felt*, pedersen_ptr:
    ↪ HashBuiltin*, range_check_ptr}(
```

```
437 func get_user_debt_for_token{syscall_ptr: felt*, pedersen_ptr:
    ↪ HashBuiltin*, range_check_ptr}(
```

CVF-45. INFO

- **Category** Readability
- **Source** functions.cairo

Recommendation As TRUE=1 and FALSE=0, the “NOT x” value could be calculated as “1 - x”.

Client Comment *Noted but won’t fix. Cairo 1 has ‘bool‘ so this should be no longer relevant.*

```
466 if (user_not_undercollateralized == TRUE) {
        return (is_undercollateralized=FALSE);
    } else {
        return (is_undercollateralized=TRUE);
470 }
```

CVF-47. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Recommendation Consider implementing a special version of the “update_rates_and_raw_total_debt” without the delta raw total debt arguments.

Client Comment *Noted but won’t fix at the moment. We tried to make internal functions as generic as possible to make it easier to expose bugs in tests, as the cost of a little bit of inefficiency as you pointed out. We prefer to not refactor these parts before launch.*

```
510 Internal.update_rates_and_raw_total_debt(
```

```
515     is_delta_raw_total_debt_negative=FALSE,
        abs_delta_raw_total_debt=0,
```

CVF-48. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description This check looks redundant. It is unclear what's wrong with a zero withdraw amount. Zero deposits are not forbidden.

Recommendation Consider just doing nothing in case "amount" is zero.

Client Comment *Noted but won't fix. We forbid zero here because internally we use zero to denote a "withdraw all" action. We understand that there's another issue stating that we should use -1 instead, but since we also won't fix that issue now, we will simply leave this one as is.*

```
540 with_attr error_message("Market: zero amount") {
        assert_not_zero(amount);
    }
```

CVF-50. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description The user is guaranteed to have debt here, so the logic of the "set_user_has_debt" function is an overcomplication here.

Recommendation Consider implementing a simplified version of the "set_user_has_debt" for such a case.

Client Comment *Noted but won't fix at the moment. We tried to make internal functions as generic as possible to make it easier to expose bugs in tests, as the cost of a little bit of inefficiency as you pointed out. We prefer to not refactor these parts before launch.*

```
581 set_user_has_debt(caller, token, raw_user_debt_before,
       ↪ raw_user_debt_after);
```

CVF-51. INFO

- **Category** Procedural
- **Source** functions.cairo

Recommendation This check should be done earlier, right after updating the protocol state.

Client Comment *Noted but won't fix until Cairo 1 rewrite, as failed transactions can't possibly be charged fees in Cairo 0, so this won't actually affect users.*

```
594 with_attr error_message("Market: insufficient collateral") {
        assert_not_undercollateralized(caller, TRUE);
    }
```

CVF-52. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description This event is emitted even if nothing actually changed.

Client Comment *Noted but won't fix. This would create a bit of noise for off-chain indexing but should be mostly fine.*

```
659 CollateralEnabled.emit(caller, token);
```

```
682 CollateralDisabled.emit(caller, token);
```

CVF-53. INFO

- **Category** Suboptimal
- **Source** functions.cairo

Description In ERC-20 the "decimals" property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts are integers.

Client Comment *Noted but won't fix. Unfortunately we have to deal with decimals at some point since we need to deal with prices from oracles, which tend to be decimal-aware. Therefore, we need to handle decimals somewhere, and we believe it's easier to do it here.*

```
703 let (debt_reserve_decimals) = reserves.read_decimals(debt_token);
```

CVF-57. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description Here, the loan fee is rounded down, i.e. towards the user. A good practice to always round towards the protocol.

Recommendation Consider rounding up.

Client Comment *Noted but won't fix. We believe the difference is negligible.*

763
```
let loan_fee = SafeDecimalMath.mul(amount, flash_loan_fee);
```

CVF-59. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description The new token balance is obtained twice. once here and another time inside the "settle_extra_reserve_balance" call.

Recommendation Consider passing the already obtained balance into the call as an argument.

Client Comment *Noted but won't fix at the moment. This was introduced as a temporary patch in https://github.com/zkLend/zklend-v1-core/pull/33 and we will try to refactor it in the Cairo 1 rewrite.*

782
```
let (reserve_balance_after_u256) = IERC20.balanceOf(
    contract_address=token, account=this_address
);
```

792
```
settle_extra_reserve_balance(token);
```

CVF-60. INFO

- **Category** Flaw
- **Source** functions.cairo

Description It is not checked that the token reserve does exists.

Recommendation Consider asserting that “reserve_index” is not zero.

Client Comment *Noted but won’t fix. In all cases the callers already checked the reserve exists. Reserve existence is not checked in these functions to avoid duplicated checks. However, we agree that it would be helpful to document this assumption and maybe changing the name to be more alarming to prevent future misuse.*

```
815 let (reserve_index) = reserve_indices.read(token);
```

```
825 let (reserve_index) = reserve_indices.read(token);
```

```
865 let (reserve_index) = reserve_indices.read(token);
```

CVF-61. INFO

- **Category** Bad naming
- **Source** functions.cairo

Description The name is confusing, as this function could be used not only to set but also to clear a flag.

Recommendation Consider renaming to “set_user_flag_state” or something like this.

Client Comment *Noted but won’t fix. This is just an internal function, but we will make sure to properly document this when rewriting the code in Cairo 1.*

```
835 func set_user_flag{
```

CVF-64. INFO

- **Category** Procedural
- **Source** functions.cairo

Description Using the “decimals” property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *Noted but won’t fix. Unfortunately we have to deal with decimals at some point since we need to deal with prices from oracles, which tend to be decimal-aware. Therefore, we need to handle decimals somewhere, and we believe it’s easier to do it here.*

1058
```
let (decimals) = reserves.read_decimals(token);
```

CVF-67. INFO

- **Category** Procedural
- **Source** functions.cairo

Recommendation This should be mode earlier, before updating accumulators.

Client Comment *Noted but won’t fix until Cairo 1 rewrite, as failed transactions can’t possibly be charged fees in Cairo 0, so this won’t actually affect users.*

1100
```
Internal.assert_reserve_enabled(token);
```

CVF-69. INFO

- **Category** Unclear behavior
- **Source** functions.cairo

Description This should be called only when “raw_user_debt_after” is zero.

Client Comment *Noted but won’t fix. We believe the way it’s written right now makes ‘repay_debt_internal()‘ cleaner.*

1191
```
set_user_has_debt(beneficiary, token, raw_user_debt_before,
    ↪ raw_user_debt_after);
```

CVF-72. INFO

- **Category** Documentation
- **Source** events.cairo

Description It is unclear what parameters this decimals value applies to.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
9 decimals: felt,
```

CVF-73. INFO

- **Category** Documentation
- **Source** events.cairo

Description The number format of these values is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
11 collateral_factor: felt,
borrow_factor: felt,
reserve_factor: felt,
```

CVF-74. INFO

- **Category** Documentation
- **Source** events.cairo

Description It is unclear whether these values are absolute of percentages. In the latter case, the number format of these values is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
14 flash_loan_fee: felt,
liquidation_bonus: felt,
```

CVF-75. INFO

- **Category** Bad naming
- **Source** events.cairo

Description The role of this user is unclear.

Recommendation Consider using a more specific name.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
50 user: felt,
```

CVF-76. INFO

- **Category** Documentation
- **Source** IZToken.cairo

Description It is unclear how a string, such as name or symbol, is packed into a felt.

Recommendation Consider documenting.

Client Comment *Won't fix. We're using native Cairo short strings (https://www.cairo-ang.org/docs/how_cairo_works/consts.html#short-string-literals) here with no custom encoding. It's inline with how popular implementations (e.g. OpenZeppelin) do it.*

```
9 func name() -> (name: felt) {
```

```
12 func symbol() -> (symbol: felt) {
```

CVF-77. INFO

- **Category** Documentation
- **Source** IZToken.cairo

Description It is unclear how a raw total supply differs from a normal total supply.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
39 func get_raw_total_supply() -> (raw_supply: felt) {
```

CVF-78. FIXED

- **Category** Unclear behavior
- **Source** IZToken.cairo

Description Unlike the “burn_all” function, this function doesn’t return the actual transferred amount.

Recommendation Consider returning it for consistency.

```
60 func transfer_all(recipient: felt) {
```

CVF-79. INFO

- **Category** Documentation
- **Source** IPriceOracleSource.cairo

Description The number format of a price is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won’t fix at the moment. Will address during the Cairo 1 rewrite.*

```
7  func get_price() -> (price: felt) {
```

```
10 func get_price_with_time() -> (price: felt, update_time: felt) {
```

CVF-80. INFO

- **Category** Documentation
- **Source** IPriceOracle.cairo

Description The number format of a price is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won’t fix at the moment. Will address during the Cairo 1 rewrite.*

```
7  func get_price(token: felt) -> (price: felt) {
```

```
10 func get_price_with_time(token: felt) -> (price: felt, update_time:
   ↪ felt) {
```

CVF-81. INFO

- **Category** Documentation
- **Source** IMarket.cairo

Description It is unclear what flags could be returned by this function.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
27 func get_user_flags(user: felt) -> (map: felt) {
```

CVF-82. INFO

- **Category** Documentation
- **Source** IMarket.cairo

Description The number format of these arguments is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
77 collateral_factor: felt,
   borrow_factor: felt,
   reserve_factor: felt,
```

CVF-83. INFO

- **Category** Documentation
- **Source** IMarket.cairo

Description It is unclear, whether these values are absolute or percentages. In the latter case, the number format of these values is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
80 flash_loan_fee: felt,
   liquidation_bonus: felt,
```

CVF-84. INFO

- **Category** Documentation
- **Source** IInterestRateModel.cairo

Description The number format of the returned values is unclear.

Recommendation Consider documenting.

Client Comment *Noted but won't fix at the moment. Will address during the Cairo 1 rewrite.*

```
8 lending_rate: felt, borrowing_rate: felt
```

CVF-85. INFO

- **Category** Documentation
- **Source** IEmpiricOracle.cairo

Description It is unclear, what are the valid aggregation modes.

Recommendation Consider documenting.

Client Comment *Won't fix. This is just a thrid-party contract interface.*

```
5 func get_value(key: felt, aggregation_mode: felt) -> (
```

CVF-86. INFO

- **Category** Suboptimal
- **Source** IEmpiricOracle.cairo

Description The "decimals" property makes the whole model more complicated, while the felt range is large enough to use the same precision for all values.

Recommendation Consider using the same precision, e.g. 18 decimals or 64 bits, and remove the "decimals" property.

Client Comment *Won't fix. This is just a thrid-party contract interface.*

```
6 value: felt, decimals: felt, last_updated_timestamp: felt,
    ↪ num_sources_aggregated: felt
```

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