Report

v. 1.0

Customer

Nomial

Smart Contract Audit

Contracts

7th April 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog **4**

2 Introduction **5**

3 Project scope **6**

4 Methodology **7**

5 Our findings **8**

6 Critical Issues **9**
CVF-1. FIXED 9

7 Major Issues **10**
CVF-2. FIXED 10
CVF-3. FIXED 10

8 Moderate Issues **11**
CVF-4. FIXED 11
CVF-5. FIXED 11

9 Recommendations **12**
CVF-7. FIXED 12
CVF-8. FIXED 12
CVF-9. FIXED 12
CVF-10. INFO 13
CVF-11. FIXED 13
CVF-12. INFO 13
CVF-13. FIXED 14
CVF-14. FIXED 14
CVF-15. FIXED 14
CVF-16. FIXED 15
CVF-17. FIXED 15
CVF-18. FIXED 16
CVF-19. FIXED 16
CVF-20. FIXED 17
CVF-21. FIXED 17
CVF-22. FIXED 17
CVF-23. FIXED 18
CVF-24. FIXED 18
CVF-25. FIXED 18
CVF-26. FIXED 19
CVF-27. FIXED 19
CVF-28. FIXED 19

CVF-29. FIXED 20
CVF-30. FIXED 20
CVF-31. FIXED 20
CVF-32. FIXED 21
CVF-33. FIXED 21
CVF-34. FIXED 21
CVF-35. FIXED 21
CVF-36. FIXED 22
CVF-37. FIXED 23
CVF-38. FIXED 23
CVF-39. FIXED 23
CVF-40. FIXED 24
CVF-41. FIXED 24
CVF-42. FIXED 24

1 Changelog

#	Date	Author	Description
0.1	07.04.25	A. Zveryanskaya	Initial Draft
0.2	07.04.25	A. Zveryanskaya	Minor revision
1.0	07.04.25	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.

Nomial is an Inventory Access Layer for solvers.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

interfaces/			
	ICollateralPool01.sol	IInventoryPool01.sol	IInventoryPool Params01.sol
/			
	CollateralPool01.sol	InventoryPool01.sol	InventoryPool Params01.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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 1 critical, 2 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Fixed 3 out of 3 issues

6 Critical Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** InventoryPool01.sol

Recommendation Should be: borrowers[borrower].penaltyDebtPaid += penaltyDebt_;

Client Comment *The logic is confusing here because of the naming and lack of comments. We renamed the variable and clarified this logic with inline comments.*

```
243 borrowers[borrower].penaltyDebtPaid = 0;
```

7 Major Issues

CVF-2 FIXED

- **Category** Flaw
- **Source** InventoryPool01.sol

Description This formula looks incorrect, as it completely ignores the current penaltyCounterStart value. So in some cases it could reduce the time until penalty, while in other cases it could actually increase that time.

Recommendation Either fix the formula to take into account the current penaltyCounterStart value, or clearly explain the reasoning behind the current logic.

Client Comment *The formula was updated to always increase penaltyCounterStart, and to do so porportionally based on the time elapsed since penaltyCounterStart was set.*

```
259 borrowers[borrower].penaltyCounterStart = block.timestamp - period_
      ↪ + paymentRatio_.mulDiv(period_, 1e27);
```

CVF-3 FIXED

- **Category** Suboptimal
- **Source** InventoryPoolParams01.sol

Description Calculating interest rate as a function of utilization rate doesn't make much sense from economical point of view. Indeed, the same approach had been taken to Aave.

Recommendation The formula should take into account the current rate and adjust it slowly: https://hackmd.io/@abdk/SyA-U6j9Je

Client Comment *We understand that you made the same suggestion to Aave. Aave V3 pools calculate interest rate as a function of utilization rate, and we are using the same formula in InventoryPoolParams01.sol. We may simplify this for some V1 pools to use an interest rate set by an owner, and we may implement your suggestion for future pools.*

```
65 function interestRate(uint utilizationRate_) external view returns (
     ↪ uint interestRate_) {
```

8 Moderate Issues

CVF-4 FIXED

- **Category** Unclear behavior
- **Source** CollateralPool01.sol

Recommendation There should be a check to ensure amount is positive, as zero amount has a special meaning of non-existing withdrawal.

```
76 function startWithdraw(IERC20 token, uint amount) public
      ↪ nonReentrant() {
```

CVF-5 FIXED

- **Category** Procedural
- **Source** InventoryPool01.sol

Description This value is calculated rounded down, which means towards the user. A good practice is to always round towards the protocol.

Recommendation Round up here.

```
83 uint scaledDebt_ = amount.mulDiv(1e27, storedAccInterestFactor) +
      ↪ amount.mulDiv(params.baseFee(), 1e27);
```

9 Recommendations

CVF-7 FIXED

- **Category** Procedural
- **Source** IInventoryPool01.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special regarding this particular version. Also relevant for:IInventoryPoolParams01.sol, ICollateralPool01.sol, CollateralPool01.sol, InventoryPool01.sol, InventoryPoolParams01.sol.

```
3 pragma solidity ^0.8.20;
```

CVF-8 FIXED

- **Category** Readability
- **Source** IInventoryPool01.sol

Description Declaring top-level errors in a file named after an interface makes it harder navigating through code.

Recommendation Move the error declarations into the interface, or move them into a separate file.

```
 7 error NotSupported();
   error Expired();
   error NoDebt();
10 error ZeroRepayment();
   error InsufficientLiquidity();
   error WrongChainId();
```

CVF-9 FIXED

- **Category** Suboptimal
- **Source** IInventoryPool01.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
12 error WrongChainId();
```

CVF-10 INFO

- **Category** Bad naming
- **Source** IInventoryPool01.sol

Recommendation Events are usually named via nouns, such as “Borrow”.

Client Comment *Past-tense verbs are not uncommon either, and make sense for most of our events. Will leave this as-is.*

```
15 event Borrowed(address indexed borrower, address indexed recipient,
      ↪ uint amount);
```

CVF-11 FIXED

- **Category** Procedural
- **Source** IInventoryPool01.sol

Description Unlike other argument names, this one has the underscore (“_”) prefix.

Recommendation Use consistent naming.

```
21 function upgrageParamsContract (address params_) external;
```

CVF-12 INFO

- **Category** Documentation
- **Source** IInventoryPool01.sol

Description The number format of the returned value is unclear.

Recommendation Explain in a documentation comment.

Client Comment *It is explained in the implementation contract. Technically does not have to be 1e27 so would not make sense to include in the interface. We’d like to keep the interfaces clean and include doc comments in implementation only.*

```
25 function utilizationRate() external view returns (uint);
```

CVF-13 FIXED

- **Category** Readability
- **Source** IInventoryPoolParams01.sol

Description Declaring a top-level error in a file named after an interface makes it harder navigating through code.

Recommendation Move the error declaration into the interface or into a separate file.

```
5 error InvalidUtilizationRate();
```

CVF-14 FIXED

- **Category** Procedural
- **Source** IInventoryPoolParams01.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
5 error InvalidUtilizationRate();
```

CVF-15 FIXED

- **Category** Procedural
- **Source** IInventoryPoolParams01.sol

Description The underscore ("_") suffix in the argument name and the return value name looks odd.

Recommendation Remove it.

```
9 function interestRate(uint utilizationRate_) external view returns (
    ↪ uint interestRate_);
```

CVF-16 FIXED

- **Category** Documentation
- **Source** IInventoryPoolParams01.sol

Description The number format for the argument and the return values is unclear.

Recommendation Explain in a documentation comment.

Client Comment *Commented in implementation contract.*

```
9  function interestRate(uint utilizationRate_) external view returns (
       ↪ uint interestRate_);
10 function penaltyRate() external view returns (uint);
```

CVF-17 FIXED

- **Category** Readability
- **Source** ICollateralPool01.sol

Description Declaring top-level errors in a file named after an interface makes it harder navigating through code.

Recommendation Move the error declarations into the interface or into a separate file.

```
7  error InsufficientBalance(uint balance);
   error NothingToWithdraw();
   error WithdrawNotReady(uint withdrawReadyTime);
10 error InsufficientLiquidity(uint amount);
   error NotSupported();
```

CVF-18 FIXED

- **Category** Bad naming
- **Source** ICollateralPool01.sol

Recommendation Events are usually named via nouns, such as “Deposit” or “WithdrawRequest”.

Client Comment *We will keep these as past-tense verbs.*

```
14 event Deposited(address indexed depositor, IERC20 token, uint amount
      ↪ );
   event WithdrawRequested(address indexed depositor, uint nonce, uint
      ↪ startTime, IERC20 token, uint amount);
   event WithdrawCompleted(address indexed depositor, uint nonce,
      ↪ IERC20 token, uint amount);
   event WithdrawPeriodUpdated(uint withdrawPeriod);
   event BalanceLiquidated(address indexed depositor, IERC20 token,
      ↪ uint amount, address recipient);
   event WithdrawLiquidated(address indexed depositor, uint nonce,
      ↪ IERC20 token, uint amount, address recipient);
```

CVF-19 FIXED

- **Category** Procedural
- **Source** ICollateralPool01.sol

Recommendation Token and nonce parameters should be indexed.

```
14 event Deposited(address indexed depositor, IERC20 token, uint amount
      ↪ );
   event WithdrawRequested(address indexed depositor, uint nonce, uint
      ↪ startTime, IERC20 token, uint amount);
   event WithdrawCompleted(address indexed depositor, uint nonce,
      ↪ IERC20 token, uint amount);
```

```
18 event BalanceLiquidated(address indexed depositor, IERC20 token,
      ↪ uint amount, address recipient);
   event WithdrawLiquidated(address indexed depositor, uint nonce,
      ↪ IERC20 token, uint amount, address recipient);
```

CVF-20 FIXED

- **Category** Procedural
- **Source** ICollateralPool01.sol

Description Unlike other argument names, this one has the underscore ("_") prefix.

Recommendation Use consistent naming.

```
26 function updateWithdrawPeriod(uint _withdrawPeriod) external;
```

CVF-21 FIXED

- **Category** Readability
- **Source** CollateralPool01.sol

Description Declaring a top-level structure in a file named after a contract makes it harder navigating through code.

Recommendation Move the declaration into the contract or move it into a separate file.

```
18 struct TokenWithdraw {
```

CVF-22 FIXED

- **Category** Suboptimal
- **Source** CollateralPool01.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are accounts and values are structs encapsulating the values of the original mappings.

```
38 mapping(address => mapping(IERC20 => uint)) public tokenBalance;
   mapping(address => mapping(uint => TokenWithdraw)) public
      ↪ tokenWithdraws;
40 mapping(address => uint) public withdrawNonce;
```

CVF-23 FIXED

- **Category** Documentation
- **Source** CollateralPool01.sol

Description The semantics of the second key is unclear.

Recommendation Explain in a documentation comment.

```
39 mapping(address => mapping(uint => TokenWithdraw)) public
   ↪ tokenWithdraws;
```

CVF-24 FIXED

- **Category** Procedural
- **Source** CollateralPool01.sol

Recommendation The “WithdrawPeriodUpdated” event should be emitted here.

```
51 withdrawPeriod = withdrawPeriod_;
```

CVF-25 FIXED

- **Category** Suboptimal
- **Source** CollateralPool01.sol

Description The value “withdrawNonce[msg.sender]” is read from the storage several times here.

Recommendation Read once and reuse.

```
82 withdrawNonce[msg.sender] += 1;
   tokenWithdraws[msg.sender][withdrawNonce[msg.sender]] =
   ↪ TokenWithdraw(token, block.timestamp, amount);
```

```
85 emit WithdrawRequested(msg.sender, withdrawNonce[msg.sender], block.
   ↪ timestamp, token, amount);
```

CVF-26 FIXED

- **Category** Suboptimal
- **Source** CollateralPool01.sol

Description The storage address for this value is already calculated and is available as "_tokenWithdraw".

Recommendation Use the already calculated storage address.

```
110 delete tokenWithdraws[msg.sender][nonce];
```

CVF-27 FIXED

- **Category** Suboptimal
- **Source** CollateralPool01.sol

Description The value of "tokenBalance[depositor][token]" is read from the storage several times.

Recommendation Read once and reuse.

```
127 if(tokenBalance[depositor][token] < amount) {
        revert InsufficientLiquidity(tokenBalance[depositor][token]);
```

```
131 tokenBalance[depositor][token] -= amount;
```

CVF-28 FIXED

- **Category** Suboptimal
- **Source** CollateralPool01.sol

Description The storage address for this value is already calculated and is available as "_tokenWithdraw".

Recommendation Use the already calculated storage address.

```
155 delete tokenWithdraws[depositor][nonce];
```

CVF-29 FIXED

- **Category** Unclear behavior
- **Source** CollateralPool01.sol

Description This event is emitted even if nothing actually changed.

```
172 emit WithdrawPeriodUpdated(_withdrawPeriod);
```

CVF-30 FIXED

- **Category** Suboptimal
- **Source** CollateralPool01.sol

Description This function is redundant, as a contract without a “receive” function anyway rejects all incoming ether transfers.

Recommendation Remove this function.

```
175 /**
     * @notice Prevents accidental ETH transfers to the contract
     * @dev Reverts any ETH transfer to the contract
     * @custom:revert NotSupported for any ETH transfer
     */
180 receive() external payable {
        revert NotSupported();
    }
```

CVF-31 FIXED

- **Category** Readability
- **Source** InventoryPool01.sol

Description Declaring a top-level structure in a file named after a contract makes it harder navigating through code.

Recommendation Move the declaration into the contract or move it into a separate file.

```
21 struct BorrowerData {
```

CVF-32 FIXED

- **Category** Documentation
- **Source** InventoryPool01.sol

Description The number format for these values is unclear.

Recommendation Explain in a documentation comment.

```
41 uint public storedAccInterestFactor;
   uint public lastAccumulatedInterestUpdate;
```

CVF-33 FIXED

- **Category** Bad datatype
- **Source** InventoryPool01.sol

Recommendation The type for these arguments should be “IInventoryPoolParams01”.

```
53 address params_
```

CVF-34 FIXED

- **Category** Bad datatype
- **Source** InventoryPool01.sol

Recommendation This address should be a named constant.

```
60 deposit(initAmount, 0x000000000000000000000000000000000000dEaD);
```

CVF-35 FIXED

- **Category** Unclear behavior
- **Source** InventoryPool01.sol

Description This should be executed only when “initAmount” is positive.

```
60 deposit(initAmount, 0x000000000000000000000000000000000000dEaD);
```

CVF-36 FIXED

- **Category** Bad datatype
- **Source** InventoryPool01.sol

Recommendation The value “1e27” should be a named constant.

```
83  uint scaledDebt_ = amount.mulDiv(1e27, storedAccInterestFactor) +
       ↪ amount.mulDiv(params.baseFee(), 1e27);

155 return totalReceivables_.mulDiv(1e27, totalAssets_);

204     return 1e27;

208         1e27 + params.interestRate(_utilizationRate(
               ↪ storedAccInterestFactor)) * (block.timestamp -
               ↪ lastAccumulatedInterestUpdate),
            1e27

258         uint paymentRatio_ = baseDebtPayment_.mulDiv(1e27, baseDebt_
               ↪ );
            borrowers[borrower].penaltyCounterStart = block.timestamp -
               ↪ period_ + paymentRatio_.mulDiv(period_, 1e27);

262     uint scaledDebt_ = baseDebtPayment_.mulDiv(1e27,
           ↪ storedAccInterestFactor, Math.Rounding.Ceil);

339 return totalReceivables_.mulDiv(1e27, totalAssets_);

349 return scaledReceivables.mulDiv(accInterestFactor, 1e27);

360 return borrowers[borrower].scaledDebt.mulDiv(accInterestFactor, 1e27
       ↪ );

374 return (_baseDebt(borrower, accInterestFactor) * penaltyTime_).
       ↪ mulDiv(params.penaltyRate(), 1e27) - borrowers[borrower].
       ↪ penaltyDebtPaid;
```

CVF-37 FIXED

- **Category** Unclear behavior
- **Source** InventoryPool01.sol

Description This function should emit some event.

```
125 function upgrageParamsContract(address params_) public onlyOwner() {
```

CVF-38 FIXED

- **Category** Bad datatype
- **Source** InventoryPool01.sol

Recommendation The argument type should be “IInventoryPoolParams01”.

```
125 function upgrageParamsContract(address params_) public onlyOwner() {
```

CVF-39 FIXED

- **Category** Suboptimal
- **Source** InventoryPool01.sol

Description This function is redundant, as a contract without a “receive” function anyway rejects all incoming ether transfers.

Recommendation Remove this function.

```
377 /**
     * @notice Fallback function to prevent accidental ETH deposits
     * @dev Reverts if ETH is transferred to the pool
380  */
    receive() external payable {
        revert NotSupported();
    }
```

CVF-40 FIXED

- **Category** Suboptimal
- **Source** InventoryPoolParams01.sol

Recommendation These variables should be declared as immutable.

```
18  uint private _baseFee;
    uint private _baseRate;
20  uint private _rate1;
    uint private _rate2;
    uint private _optimalUtilizationRate;
    uint private _penaltyRate;
    uint private _penaltyPeriod;
```

CVF-41 FIXED

- **Category** Documentation
- **Source** InventoryPoolParams01.sol

Description The number format for these variables is unclear.

Recommendation Explain in a documentation comment.

```
19  uint private _baseRate;
20  uint private _rate1;
    uint private _rate2;
    uint private _optimalUtilizationRate;
    uint private _penaltyRate;
```

CVF-42 FIXED

- **Category** Documentation
- **Source** InventoryPoolParams01.sol

Description The number format for these arguments is unclear.

Recommendation Explain in a documentation comment.

```
28  uint baseFee_,
    uint baseRate_,
30  uint rate1_,
    uint rate2_,
    uint optimalUtilizationRate_,
    uint penaltyRate_,
```

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