HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Bit5 SCRA
Date: 11 July, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Bit5 SCRA
Approved By	Oleksii Zaiats | Head of SC Audits at Hacken OU
Type	Marketplace; Lending Platform
Platform	Binance Smart Chain, Ethereum, Avalanche
Language	Solidity
Methodology	Link
Website	https://bit5.com/
Changelog	23.05.2023 - Initial Review 19.06.2023 - Second Review 11.07.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Bit5 SCRA (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Bit5 SCRA is a NFT Marketplace and NFT Lending Platform system with the following contracts:

- *Bit5* — is a contract with the tokens marketplace functionality. During the contract initialization, the fee is set to 2% and the maximal royalty is set to 40%. The contract has a pausing functionality: it is not possible to buy tokens, accept bids or cancel the order.

  Orders used in the contract are created off-chain. The issuer is required to sign the order information. In the contract, those who wish to accept the order provide the order information and the issuer's signature. These data are validated, checking if the signer matches the issuer specified in the order information. Additionally, the validation ensures that the order is not expired, using the *end* value in the order, and verifies if the payment ERC-20 token is allowed.

  Once validated, the order is processed. The order can be either a LISTING or a BID, and the contract has separate functions for each type. In the case of a BID, msg.sender sells the NFT to the order issuer, while in a LISTING, it is the opposite.

  The service fee is collected in ERC-20 payment tokens. From the service fee, the treasury fee is deducted and sent to the *Bit5Treasury* contract. If the NFT supports royalties, the royalty for the token is sent. Otherwise, custom royalties can be sent. The collection owner can set collection royalties with a percentage share, with the total percentage not exceeding 40%.

  Additionally, the issuer can be the owner of a privileged collection. The contract owner can establish privileged collections along with their respective percentages. In such cases, the service fee is reduced by the privilege percentage, and the treasury fee is adjusted accordingly. After this, the payment tokens are sent to the seller, and the NFT is transferred to the buyer.

  Furthermore, the contract allows processing global BIDs. In the order information, the issuer specifies the quantity of tokens they wish to purchase from the collection, and users can sell them the corresponding quantity of any tokens from the collection.

- *Bit5Lending* – is a contract with the tokens borrowing and lending functionality.

  Orders are created and processed in the same manner as in the previous contract. The order owner has the ability to cancel their own order. Similar to the previous contract, an order can have either an OFFER (where the issuer wants to provide a loan) or a LIST (where the issuer wants to lend) type. The order object specifies an array of NFT addresses, their IDs, quantities, and types (ERC721 or ERC1155). It also includes the signingTime and expiration (the order becomes invalid after signingTime + expiration), as well as the duration (the timeframe within which the borrower must repay the debt) which is indicated in the order information.

  When the borrower repays the debt, they also pay the interestRate, which is specified in the order information. If the borrower fails to repay the debt, the lender who provided the loan can liquidate it (collateral NFTs are transferred to them), and the collaterals are sent to the contract at the time of order acceptance. Additionally, a loan order can be global, meaning the lender can provide a loan against any NFT from the specified collection.

  Upon loan initiation, a service fee is collected from the loan (initially set at 1% during contract initialization but can be modified by the owner). The contract also includes pausing functionality, which allows for the blocking of order cancellations, acceptance, payment, and liquidation operations.
- *Bit5Treasury* – is a contract that facilitates the collection of treasury fees from the contract, stores balances associated with an NFT collection, and allows for their withdrawal.
- *Create3Factory* – is a contract that provides the functionality for the contracts deployment.
- *TransparentProxy* – is a transparent upgradeable proxy contract.
- *LibOrder.sol* – provides structs and enums for the *Bit5* and the *LibOrder* library with the order hashing function.
- *LibOrderLending.sol* – provides structs and enums for the *Bit5* and the *LibOrderLending* library with the lending order hashing function.
- *OrderValidator* – is a contract that provides a function of obtaining the signer address from the order and signature.
- *OrderValidatorLending* – is a contract that provides a function of obtaining the signer address from the lending order and signature.
- *Bit5Errors* – is an interface that provides errors used in the *Bit5* and *Bit5Treasury* contracts.
- *IBit5Treasury* – is an interface that defines *Bit5Treasury* contract functions.

- *solmate/src/utils/CREATE3* – is a contract that provides the functionality for the contracts deployment.
- *solmate/src/utils/Bytes32AddressLib – is a* library for converting addresses to bytes32 and vice versa.

Privileged roles

- The owner of the *Bit5* contract can withdraw native coin and ERC-20 tokens from the contract, allow and disallow payment tokens, pause and unpause the contract, change service fee, set privileged NFT collections with percentages, and change treasury fee percentages.
- The owner of the *Bit5* contract can acceptGlobalBidAsOwner on behalf of any user.
- The owner of the *Bit5Lending* contract can withdraw native coin and ERC-20 tokens from the contract, allow and disallow payment tokens, pause and unpause the contract, change the service fee, set payment tokens statuses.
- The owner of the *Bit5Treasury* contract can set *bit5* address and withdraw fees from the contract.
- The *bit5* address in the *Bit5Treasury* contract can call a *deposit* function (that transfers treasury fee to this contract.)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are provided and are sufficient.
- The technical description is limited:
  - The technical specification is not provided.
  - NatSpec is missing.

Code quality

The total Code Quality score is **8** out of **10**.

- Insufficient Gas modeling.
- Solidity Style Guide violations (code formatting, naming conventions).

Test coverage

Code coverage of the project is **94.35%** (branch coverage).

Security score

As a result of the audit, the code contains **1** low severity issue. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.2**. The system users should acknowledge all the risks summed up in the risks section of the report.

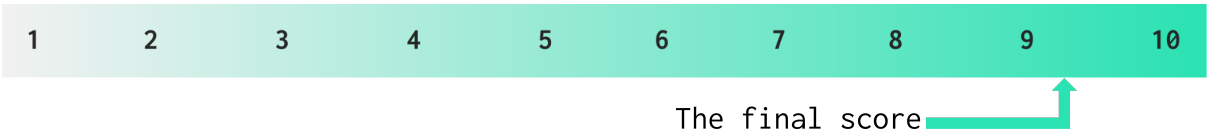

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
23 May 2023	4	2	13	1

19 June 2023	6	3	8	0
11 July 2023	1	0	0	0

Risks

- The contracts are upgradeable and may be modified.
- The service fees are not limited and may be changed by the contracts` owners.
- Highly permissive roles are present

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Failed	L01
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Not Relevant	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	
Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Failed	I08
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Data Consistency

Impact	High
Likelihood	Medium

The offer kind (*BID* or *LIST*) is not verified when calling the functions for the orders processing.

Therefore, the orders may be processed in an incorrect way, which may lead to incorrect tokens transferals or manipulations.

Path:
./contracts/Bit5.sol : buy(), acceptBid(), acceptGlobalBid(), acceptGlobalBidAsOwner();

Recommendation: verify if the order kind is appropriate when processing it.

Found in: 0e43cbc

Status: Fixed (Revised commit: b6f2142)

■■■ High

H01. Undocumented Behavior

Impact	High
Likelihood	Medium

The contracts have a pausing functionality.

Users should be acknowledged about this behavior.

Paths:
./contracts/Bit5.sol : pause(), unpause();
./contracts/Bit5Lending.sol : pause(), unpause();
./contracts/Bit5Treasury.sol : pause(), unpause();

Recommendation: describe the pausing functionality in the documentation.

Found in: 0e43cbc

Status: Mitigated (Revised commit: 196e206) (The behavior is documented)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

H02. Denial of Service Vulnerability

Impact	Medium
Likelihood	High

When adding new payment tokens, they are not approved to the *Bit5_TREASURY*.

This will lead to the inability of new token usage because the transactions will fail in the *_doTransfers* function when calling the *Bit5_TREASURY.deposit*.

Path:
./contracts/Bit5.sol : setPaymentToken();

Recommendation: approve the newly added tokens to the *Bit5_TREASURY* and set the allowance to 0 when removing the token.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

H03. Requirements Violation; Data Consistency

Impact	High
Likelihood	Medium

The global bid order may have the *tokenId* value set, though the global bid should not have it and will skip this value when processing the order in the *acceptGlobalBid* and *acceptGlobalBidAsOwner* function.

Additionally, the global bid order may be processed using *buy* and *acceptBid* functions intended to process *LIST* order and not global *BID* respectively.

This is a consistency violation and may lead to the incorrect input data processing, which may result in unexpected ways for the users.

Path:
./contracts/Bit5.sol : buy(), acceptBid(), acceptGlobalBid(), acceptGlobalBidAsOwner();

Recommendation: split global and usual BIDs functionality.

Found in: 0e43cbc

Status: Fixed (Revised commit: b6f2142)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

H04. Denial of Service Vulnerability

Impact	**High**
Likelihood	Medium

It is not allowed to cancel the order if the payment token is not allowed. (*cancelOrder* function checks *paymentTokens* whitelist in *isOrderValid* modifier).

Due to this, if the owner disables tokens for listed items, it will result in the impossibility to cancel the order.

Path:
./contracts/Bit5.sol : cancelOrder();

Recommendation: allow cancel orders when the payment token is no longer valid.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

H05. Data Consistency

Impact	**High**
Likelihood	Medium

Order struct has no unique per listing fields, so signature collision is possible. This leads to the impossibility of new order creation because *orderStatus* is cached per signature.

For example: The user lists their NFT, then cancels the order. *orderStatus* is updated to canceled. Then the user lists the same NFT again (using *end* variable the same as previous one), so the signature is the same as in the previous listing. Other users could not buy a listed item, as *orderStatus* is *CANCELED*, so the *_checkOrderValidity* function reverts the transaction.

Path:
./contracts/Bit5.sol : cancelOrder();

Recommendation: make the signature for each order unique: add the unique value to the hash or solve the collision problem in another way.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

H06. Requirements Violation

Impact	Medium
Likelihood	High

According to the documentation, the collateral tokens should be allowed in terms of security:

- *Your NFT must be listed on Bit5.*
- *Your NFT must not be removed from Bit5 or registered as stolen in the BNB blockchain.*
- *If your NFT meets these requirements, you can use it as collateral to borrow BNB.*

However, validations for the collaterals are not implemented.

Path:
./contracts/Bit5Lending.sol;

Recommendation: implement the code according to the requirements.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

H07. Unlimited Fees; Undocumented Behavior

Impact	High
Likelihood	Medium

The fees are not limited and may exceed 100%.

All the fees and limits should be documented and users should be acknowledged on the fee amounts and limits.

In case the fees exceed 100%, the calculations will not work properly: *_doTransfers* function could be reverted because of insufficient balance in multiple scenarios.

Scenario 1 (2% standard fee, 5% royalty fee, treasuryPercentages is 0, not a privileged collection): user buys an NFT for 100 tokens, standardServiceFee is 2, treasuryFeePercentage is 0. After paying the royalty **contract has 95 tokens**. *order.price (100) - standardServiceFee (2) - totalRoyaltyAmount (5) + privilegedServiceFee (0) =* ***93*** **tokens** sent to seller. Result: contract has 2 tokens as a leftover of the operation.

Scenario 2 (2% standard fee, 5% royalty fee, treasuryPercentages is 30_000, not a privileged collection): user buys an NFT for 100 tokens, standardServiceFee is 2, toCollectionTreasury is 6. After paying the royalty contract has 95 tokens. After sending tokens to

the treasury, the contract **has 89 tokens**. *order.price (100) - standardServiceFee (2) - totalRoyaltyAmount (5) + privilegedServiceFee (0) = **93*** **tokens** sent to seller. Result: transaction is reverted due to low balance.

Scenario 3 (2% standard fee, 5% royalty fee, treasuryPercentages is 30_000, privileged collection with a value of 30_000): user buy an NFT for 100 tokens, standardServiceFee is 2, toCollectionTreasury is 6. After paying the royalty contract has 95 tokens. After sending tokens to the treasury, the contract has 89 tokens. new FEE is 200 * 30_000 / 10_000 = 600. privilegedServiceFee is 6 tokens. treasuryFeePercentage is 30_000 * 600 / 10_000 = 1800. 18 tokens sent to the treasury, **balance is 77**. *order.price (100) - standardServiceFee (2) - totalRoyaltyAmount (5) + privilegedServiceFee (6) = **99*** **tokens** sent to seller. Result: transaction is reverted due to low balance.

Because fees are not limited - the situation is possible when the seller would receive nothing.

Paths:
./contracts/Bit5.sol : _doTransfers(), changeTreasuryPercentage(), changeFee(), initialize(), changePrivilegePercentage();
./contracts/Bit5Lending.sol : changeFee(), initialize();

Recommendation: limit max fees paid during transfers and document them. Check calculation to prevent transaction revert due to low balance.

Found in: 0e43cbc

Status: Mitigated (Revised commit: 196e206) (Such behavior is a part of business logic as per client's requirements)

H08. Data Consistency

Impact	**High**
Likelihood	Medium

The global orders are not properly split with the usual ones and may be processed vice versa, as well as the orders with *OFFER* and *LIST* types.

The *Order* with *isGlobal* *true* value may have *nftAddresses*, *colleteralTokenIDs* and *amounts* values, which should not be allowed for global order.

It is possible to create a global *LIST* order, as such verification is missed in the *acceptGlobalOffer* function. In case the order with *LIST* type is processed using the *acceptGlobalOffer*, it may be impossible to payback and liquidate it, as the global token ids are set to the

order in the *payback* and *liquidate* functions only in case *orderKind* is *OFFER*. Due to this, the NFTs will be locked in the contract and the borrower will not get any tokens back.

The amount of ERC-1155 tokens to be transferred can not be set when accepting the global order. Due to this, the incorrect data may be processed.

This is a consistency violation and may lead to the incorrect input data processing, which may result in unexpected ways for the users.

Path:
./contracts/Bit5Lending.sol : processOrder(), acceptGlobalOffer(), payback(), liquidate();

Recommendation: split the global and usual orders, verify the order kind.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

H09. Data Consistency; Assets Integrity

Impact	**High**
Likelihood	Medium

The order expiration time (*order.times.signingTime + order.times.expiration*) is not checked when the global offer is accepted in the *acceptGlobalOffer* function.

This may result in an invalid order processing.

Path:
./contracts/Bit5Lending.sol : acceptGlobalOffer();

Recommendation: validate the order deadline in the *acceptGlobalOffer* function.

Found in: 0e43cbc

Status: Fixed (Revised commit: b6f2142)

H10. Undocumented Behavior

Impact	**High**
Likelihood	Medium

The global BID (*Bit5*) and offer (*Bit5Lending*) allow the transfer of any NFT from the collection to the issuer.

Such functionality is not described in the documentation, and users may not be acknowledged for such behavior and get the tokens that were not expected.

Paths:
./contracts/Bit5Lending.sol : acceptGlobalOffer();
./contracts/Bit5.sol : acceptGlobalBid(), acceptGlobalBidAsOwner();

Recommendation: describe the global orders in the documentation.

Found in: 0e43cbc

Status: Mitigated (Revised commit: 196e206) (Such behavior is a part of business logic as per client's requirements)

H11. Assets Integrity; Highly Permissive Role

Impact	High
Likelihood	Medium

Owner is allowed to pass the NFT owner's address (*_nftOwner*) in the *acceptGlobalBidAsOwner* function. After that, the NFT tokens are sent from the NFT owner's address to the *order.issuer*.

In case the owner key leaks, after the NFT owner approves the NFT contract for accepting BID, an attacker can call *acceptGlobalBidAsOwner* with the owner's address, manipulate the price, and steal the token.

Path:
./contracts/Bit5.sol : acceptGlobalBidAsOwner();

Recommendation: do not allow passing the NFT owner's address.

Found in: 0e43cbc

Status: Mitigated (Revised commit: 196e206) (Such behavior is a part of business logic as per the client's requirements and limited to the owner only. Client will use multisig account to protect keys from the leak.)

H12. Data Consistency

Impact	Medium
Likelihood	High

The *Bit5Treasury* contract allows to deposit any ERC-20 token for a specific collection address, but *collectionBalances* are updated without mapping to the ERC20 token.

As a result, the incorrect calculation of the balances and incorrect tokens may be withdrawn through the withdraw function.

Path:
./contracts/Bit5Treasury.sol : deposit(), withdraw();

Recommendation: consider creating collectionAddress -> tokenAddress -> amount mapping.

Found in: 0e43cbc

Status: Fixed (Revised commit: b6f2142)

H13. Denial Of Service Vulnerability

Impact	Low
Likelihood	Medium

addRoyaltier, *deleteRoyaltier* and functions iterate over all the royalties, which are not limited.

It is possible to add 4000 royalties with 1 basis point per user (maxRoyalty is limited to 4000), which will lead to transaction revert during looping because of Gas limit.

Path:
./contracts/Bit.sol : addRoyaltier(), deleteRoyaltier(), _doTransfers();

Recommendation: limit the max amount of royalties.

Found in: 0e43cbc

Status: Fixed (Revised commit: b6f2142)

Medium

M01. Unchecked Transfers

Impact	Low
Likelihood	Medium

The contracts do not use the *SafeERC20* library for checking the result of the ERC20 token transfer.

Tokens may not follow the ERC20 standard and return *false* in case of transfer failure or not returning any value at all. This will lead to incorrect data processing and processing the orders when the tokens are not transferred.

Paths:
./contracts/Bit5Lending.sol : _processOrder(), payback();

./contracts/Bit5Treasury.sol : deposit(), withdraw();
./contracts/Bit5.sol : _doTransfers();

Recommendation: use the *SafeERC20* library to conduct transfers.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

M02. CEI Pattern Violation

Impact	Low
Likelihood	Low

The Checks-Effects-Interactions pattern is violated. During the *deposit* function, *collectionBalances[collectionAddress]* state variable is updated after the *transferFrom* external calls.

Path:
./contracts/Bit5Treasury.sol : deposit();

Recommendation: follow the CEI pattern.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

M03. Tests Failing

Impact	Medium
Likelihood	Medium

The *Accept global bid: ERC721: accept global bid as owner* and *Treasury : should approve token* tests are failing.

This is a consistency violation and may lead to the incorrect input data processing, which may result in unexpected ways for the users.

Path:
./contracts/Bit5.sol : buy(), acceptBid(), acceptGlobalBid(), acceptGlobalBidAsOwner();

Recommendation: split global and usual BIDs structs.

Found in: b6f2142

Status: Fixed (Revised commit: 196e206)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Low

L01. Floating Pragma

Impact	Low
Likelihood	Low

Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:
./contracts/*

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Found in: 0e43cbc

Status: Reported

L02. Contradiction; Redundant Code

Impact	Low
Likelihood	Low

Bit5Treasury implements *PausableUpgradeable* contract and functions to *pause* and *unpause* functionality.

However, no functions from this contract use pausable modifiers, which makes this logic redundant or that the pausing requirements are violated.

Path:
./contracts/Bit5Treasury.sol : pause(), unpause();

Recommendation: implement missing logic, or remove redundant code.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

L03. Missing Events Emitting

Impact	Low
Likelihood	Low

The event is not emitted when setting the *bit5* address.

The state changes should be conducted together with the corresponding events emitting to track them off-chain.

Paths:
./contracts/Bit5Treasury.sol : setBit5();
./contracts/Bit5.sol : initialize();
./contracts/Bit5Lending.sol : initialize();

Recommendation: create the corresponding event and emit it.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

L04. Best Practice Violation

Impact	Low
Likelihood	Low

After the `_doTransfers` is called in the `Bit5._acceptBid` and `Bit5._acceptGlobalBidAsOwner` functions, the state variables are updated.

The token transfers are performed before the state variables are updated in the `Bit5Lending._processOrder` function.

The state variables are updated after the `_transferNFT` function is called in the liquidate and payback functions.

Paths:
./contracts/Bit5.sol : _acceptBid(), _acceptGlobalBidAsOwner(), _globalAcceptBid();
./contracts/Bit5Lending.sol : _processOrder(), acceptGlobalOffer(), payback(), liquidate()

Recommendation: follow the CEI pattern.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

L05. Incorrect Verifications

Impact	Low
Likelihood	Low

The `orderKind` checks are incorrect: if the `order.orderKind` is `OrderKind.OFFER`, it may not be `OrderKind.LIST`, therefore, the `order.orderKind == OrderKind.LIST` verification is redundant.

If the *order.orderKind* is *OrderKind.LIST*, it may not be *OrderKind.OFFER*, therefore, the *order.orderKind != OrderKind.LIST* verification is redundant.

Path:
./contracts/Bit5Lending.sol : acceptGlobalOffer()

Recommendation: fix the verifications.

Found in: b6f2142

Status: Fixed (Revised commit: 196e206)

L06. Data Consistency

Impact	Low
Likelihood	Low

The global bid order may have the *tokenId* value set, though the global bid should not have it and will skip this value when processing the order.

This is a consistency violation.

Path:
./contracts/Bit5Lending.sol : acceptGlobalOffer()

Recommendation: fix the verifications.

Found in: b6f2142

Status: Fixed (Revised commit: 196e206)

Informational

I01. Duplicated Code

The *acceptGlobalBid* and *acceptGlobalBidAsOwner* functions have a lot of common code.

Code duplication is a violation of best coding practices, and it decreases code readability.

Path:
./contracts/Bit5.sol : acceptGlobalBid(), acceptGlobalBidAsOwner();

Recommendation: consider moving the common code into separate functions and reuse it.

Found in: 0e43cbc

Status: Mitigated (Revised commit: 196e206) (In the latest code, functions are not the same)

I02. Duplicated Code

The *if (msg.sender == order.issuer)* verification is duplicated in the functions for order processing.

Code duplication is a violation of best coding practices, and it decreases code readability.

Path:
./contracts/Bit5.sol : acceptGlobalBidAsOwner(), buy(), acceptBid(), acceptGlobalBid();

Recommendation: consider moving the common code into separate functions and reuse it.

Found in: 0e43cbc

Status: Reported

I03. Not Indexed Parameters In Events

The contracts contain events with no indexed parameters.

It is recommended to index the parameters for better tracking.

Paths:
./contracts/Bit5.sol;
./contracts/Bit5Lending.sol;

Recommendation: consider adding *indexed* keyword to the important parameters in the events.

Found in: 0e43cbc

Status: Reported

I04. Inefficient Gas Model

The *address(Bit5_TREASURY)* is used, when the *treasury* can be used.

This will result in less Gas usage.

Path:
./contracts/Bit5.sol : initialize();

Recommendation: use the *treasury* value instead of the *address(Bit5_TREASURY)*.

Found in: 0e43cbc

Status: Reported

I05. Public Functions That Could Be Declared External

There are public functions in the contracts that are never called inside.

The usage of external visibility requires less Gas.

Paths:
./contracts/Bit5.sol : initialize(), pause(), unpause(), cancelOrder(), buy(), acceptBid(), acceptGlobalBid(), acceptGlobalBidAsOwner();
./contracts/Bit5Treasury.sol : initialize(), pause(), unpause();
./contracts/Bit5Lending.sol : initialize(), pause(), unpause(), cancelOrder();

Recommendation: change the visibility of the functions that are never called inside the contracts to external.

Found in: 0e43cbc

Status: Reported

I06. Unused Errors

There are errors in the project that are never used.

Redundant code decreases the readability.

Paths:
./contracts/Bit5Lending.sol : AlreadyProcessed(), InvalidOperation();
./Errors.sol : WrongOrderKind(), WrongTokenKind(), TokenTransferFailed();

Recommendation: remove the unused errors.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

I07. Unused Events

The `WithdrawRaw` event is never used.

Redundant code decreases the readability.

Path:
./contracts/Bit5Treasury.sol : WithdrawRaw();

Recommendation: remove the unused event.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

I08. Style Guide Violations

The provided projects should follow the official guidelines:

- Only constants should be named with all capital letters with underscores separating words.

- Order of layout, order of functions and modifiers should be followed.

Paths:
./contracts/Bit5Lending.sol : FEE;
./contracts/Bit5.sol : FEE;

Recommendation: fix style guide violations.

Found in: 0e43cbc

Status: Reported

I09. Redundant Operation

Bit5Lending and *Bit5* contracts have the option to withdraw native tokens; however, there is no payable function to send native tokens to the contract.

Paths:
./contracts/Bit5Lending.sol : withdrawBNB();
./contracts/Bit5.sol : withdrawBNB();

Recommendation: remove the redundant functions.

Found in: 0e43cbc

Status: Reported

I10. Inefficient Gas Model

_doTransfers function in case of privileged collection writes storage twice (update *FEE*), but this action is redundant and is not Gas efficient.

Path:
./contracts/Bit5.sol : _doTransfers();

Recommendation: do not update the storage variable. Use a local variable instead.

Found in: 0e43cbc

Status: Reported

I11. Code Consistency

The project uses *if* statements for the verifications, but the *require* statements are used in several places.

The code style should be consistent.

Paths:
./contracts/Bit5.sol : cancelOrder(), _acceptGlobalBidAsOwner(), buy(), acceptBid(), acceptGlobalBid(), acceptGlobalBidAsOwner(), addRoyaltier(), _doTransfers();

./contracts/Bit5Lending.sol : cancelOrder()

Recommendation: follow the same code style through the project.

Found in: 0e43cbc

Status: Reported

I12. Typos In The Code

There are typos in the contracts.

They decrease the code readability.

Paths:
./contracts/Bit5Lending.sol : event Liquidated - liquidater;
./contracts/libraries/LibOrderLending.sol : struct Order - colleteralTokenIDs;

Recommendation: fix the typos.

Found in: 0e43cbc

Status: Reported

I13. Commented Code

`if (!res) revert TokenTransferFailed();` line is commented.

Commented code decreases the code readability.

Path:
./contracts/Bit5Treasury.sol : deposit()

Recommendation: remove the redundant commented code.

Found in: 0e43cbc

Status: Fixed (Revised commit: b6f2142)

I14. Missing Zero Address Validation

It is not checked if the `_bit5` value is not zero address.

This may lead to unexpected zero address interaction.

Path:
./contracts/Bit5Treasury.sol : setBit5()

Recommendation: verify if the `_bit5` value is not zero address.

Found in: 0e43cbc

Status: Fixed (Revised commit: 196e206)

I15. Unused Functions

The *bytes32ToBytes* function is not used.

The redundant code decreases the code readability.

Path:
./contracts/Bit5.sol : bytes32ToBytes()

Recommendation: remove the redundant code.

Found in: b6f2142

Status: Fixed (Revised commit: 196e206)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/Bit5Tech/Bit5Contracts/
Commit	0e43cbc82f86acd73e8fe6edecc4268b3c376ce1
Whitepaper	Not provided.
Requirements	https://docs.bit5.com/
Technical Requirements	https://docs.bit5.com/
Contracts	File: ./contracts/Bit5.sol SHA3: e731db6a4c667a2dced9fe0c7895f568e9d5348106552222b655f02f302ec994 File: ./contracts/Bit5Lending.sol SHA3: a0d0369d374529db05d0a9b54bbdd878a2d53c8dd4e135c8dcb2f2dcef6dce65 File: ./contracts/Bit5Treasury.sol SHA3: 09872855cb9693ffdc4609c6e93bec3de7a271b3a23b0c6ae0cbf7f9f48cb4c4 File: ./contracts/Create3Factory.sol SHA3: 70c7a21d386df6dc8768c94b75b457ced6e711e50a5d6f2bd930350a0860aa0d File: ./contracts/TransparentProxy.sol SHA3: dd2c200cfe1626f9ee2d9f5915ae5b82a795e17a9cd7548069b8c84fbbbd7b21 File: ./contracts/interfaces/Errors.sol SHA3: 6eb084a4699940504a99b17e0e26846e831dda1e1f0f819da2459a423e164efa File: ./contracts/interfaces/IBit5Treasury.sol SHA3: df88bcfb62af47464549d6408c8227d71d0e6537d36b3933d974eee50890dfd6 File: ./contracts/libraries/LibOrder.sol SHA3: 8235015cbfb92dd13199b74a159c89deca144b543b91b59cfefef9d0393e3c74 File: ./contracts/libraries/LibOrderLending.sol SHA3: cc6d772ffa45b60d54e77a769b98f023069e8415af4c30b9f25820ea4c17f66d File: ./contracts/libraries/OrderValidator.sol SHA3: 0f1bd18e1fe5f362970a8e28325cdcbaca68579ddd520ba898c2053a4dcfc89d File: ./contracts/libraries/OrderValidatorLending.sol SHA3: f05722459738e2407d2e521a6558a9658bc884429ca61e3d6192fa78e06e2748 File: ./src/utils/Bytes32AddressLib.sol (solmate) SHA3: 2d5a9345be6b6062af95c1d42295c8001ae2dd0e444e79ca770218a52633b226 File: ./src/utils/CREATE3.sol (solmate) SHA3: a0bbc3478c96cfbe03496c2dfaffde6165b65a40448165a40bbee07df90eedbe

Second review scope

Repository	https://github.com/Bit5Tech/Bit5Contracts/
Commit	b6f214277bfa807ac7fffc8bf825781bcf06e517
Whitepaper	Not provided.
Requirements	https://docs.bit5.com/
Technical Requirements	https://docs.bit5.com/
Contracts	File: ./contracts/Bit5.sol SHA3: 1d3d57ef8ad22d76dd44a5f9b9da5aaa649dd6e10abf00dbba467475ff980671 File: ./contracts/Bit5Lending.sol SHA3: 0f7257f587e1c375cf258fb143ef9958304aa7beb4bf92b6a8b566853085f8c9 File: ./contracts/Bit5Treasury.sol SHA3: 5e4f76a393974a376ecfdb5d665b7fed21ac0f8699a222972ab52f3cc13ff696 File: ./contracts/Create3Factory.sol SHA3: 70c7a21d386df6dc8768c94b75b457ced6e711e50a5d6f2bd930350a0860aa0d File: ./contracts/TransparentProxy.sol SHA3: dd2c200cfe1626f9ee2d9f5915ae5b82a795e17a9cd7548069b8c84fbbbd7b21 File: ./contracts/interfaces/Errors.sol SHA3: 6eb084a4699940504a99b17e0e26846e831dda1e1f0f819da2459a423e164efa File: ./contracts/interfaces/IBit5Treasury.sol SHA3: cb061945decc3893db83f8cd870d743fd669c9460768a745343439cd8765c029 File: ./contracts/libraries/LibOrder.sol SHA3: 8235015cbfb92dd13199b74a159c89deca144b543b91b59cfefef9d0393e3c74 File: ./contracts/libraries/LibOrderLending.sol SHA3: cc6d772ffa45b60d54e77a769b98f023069e8415af4c30b9f25820ea4c17f66d File: ./contracts/libraries/OrderValidator.sol SHA3: 0f1bd18e1fe5f362970a8e28325cdcbaca68579ddd520ba898c2053a4dcfc89d File: ./contracts/libraries/OrderValidatorLending.sol SHA3: f05722459738e2407d2e521a6558a9658bc884429ca61e3d6192fa78e06e2748 File: ./src/utils/Bytes32AddressLib.sol (solmate) SHA3: 2d5a9345be6b6062af95c1d42295c8001ae2dd0e444e79ca770218a52633b226 File: ./src/utils/CREATE3.sol (solmate) SHA3: a0bbc3478c96cfbe03496c2dfaffde6165b65a40448165a40bbee07df90eedbe

Third review scope

Repository	https://github.com/Bit5Tech/Bit5Contracts/
Commit	196e2061ad8cc3d77fc546a3dd5491032c7aaeb3

Whitepaper	Not provided.
Requirements	https://docs.bit5.com/
Technical Requirements	https://docs.bit5.com/
Contracts	File: ./contracts/Bit5.sol SHA3: b87a61b3aa6fde92c4f9ab680162babee00b2e8a084ec916132a4be21439818a File: ./contracts/Bit5Lending.sol SHA3: c564a774f7b0b616a1707b44046e95b47c4bbe9c5e9cfc12d5cba4730a568b6f File: ./contracts/Bit5Treasury.sol SHA3: 3b8c985a099c2f4bb2860d4318388cf2a4ecbbab241268fb544a1f62793b1dd1 File: ./contracts/Create3Factory.sol SHA3: 70c7a21d386df6dc8768c94b75b457ced6e711e50a5d6f2bd930350a0860aa0d File: ./contracts/TransparentProxy.sol SHA3: 965a99132798b3a5e3da10c357093cfd53c6dfedd4780ea329052a61a3d89f41 File: ./contracts/interfaces/Errors.sol SHA3: f2daa802e0698246ec4006c4359fbe1128c8187a5d0f04309d6fa64ed31bb57c File: ./contracts/interfaces/IBit5Treasury.sol SHA3: df88bcfb62af47464549d6408c8227d71d0e6537d36b3933d974eee50890dfd6 File: ./contracts/libraries/LibOrder.sol SHA3: 8235015cbfb92dd13199b74a159c89deca144b543b91b59cfefef9d0393e3c74 File: ./contracts/libraries/LibOrderLending.sol SHA3: cc6d772ffa45b60d54e77a769b98f023069e8415af4c30b9f25820ea4c17f66d File: ./contracts/libraries/OrderValidator.sol SHA3: 0f1bd18e1fe5f362970a8e28325cdcbaca68579ddd520ba898c2053a4dcfc89d File: ./contracts/libraries/OrderValidatorLending.sol SHA3: f05722459738e2407d2e521a6558a9658bc884429ca61e3d6192fa78e06e2748 File: ./src/utils/Bytes32AddressLib.sol (solmate) SHA3: 2d5a9345be6b6062af95c1d42295c8001ae2dd0e444e79ca770218a52633b226 File: ./src/utils/CREATE3.sol (solmate) SHA3: a0bbc3478c96cfbe03496c2dfaffde6165b65a40448165a40bbee07df90eedbe