HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Colexion
Date: December 10th, 2021

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 Colexion.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC721 token
Platform	Polygon / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Filename	2ndAuditFixedColexion.sol
md5 hash	4D687A8CF46AAC2613703D6F89EECCF7
Deployed Contract	https://polygonscan.com/address/0xa5bba3d9465b003ab6fe59e42fe582aefb30c50d#readContract
Technical Documentation	YES
JS tests	NO
Website	colexion.io
Timeline	13 OCTOBER 2021 - 10 DECEMBER 2021
Changelog	19 OCTOBER 2021 - INITIAL AUDIT 12 NOVEMBER 2021 - SECOND REVIEW 23 NOVEMBER 2021 - THIRD REVIEW 10 DECEMBER 2021 - FOURTH 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

Introduction

Hacken OÜ (Consultant) was contracted by Colexion (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 contract and its code review conducted between October 13th, 2021 - October 19th, 2021.

Second review conducted on November 12th, 2021.

Third review conducted on November 23rd, 2021.

Fouth review conducted on December 10th, 2021.

Scope

The scope of the project is smart contracts in solidity file:

Filename:
2ndAuditFixedColexion.sol
md5 hash:
4d687a8cf46aac2613703d6f89eeccf7
Deployed Contract:
https://polygonscan.com/address/0xa5bba3d9465b003ab6fe59e42fe582aefb30c50d#readContract
Technical Documentation: Yes (Provided: Smart Contract Description.pdf)
JS tests: No
Contracts:
Colexion

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

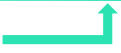
Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level

	▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.

Insecure	Poor secured	Secured	Well-secured

You are here 

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **1** critical, **2** high and **7** low severity issues.

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

After the second review security engineers found **1** high and **3** low severity issues.

After the third review security engineers found that all previously found issues were resolved, but new ones were discovered. Therefore there are **1** high and **1** low severity issue.

After the fourth review security engineers found that **all issues** were addressed.

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

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

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

Audit overview

Critical

1. Reentrancy

There is a possibility to re-enter the function for the message sender. If the transaction would be created from the contract which does have a fallback function it will be able to reenter the function again and again, until not drain all balance of the Colexion contract

Contracts: Colexion

Functions: withdraw, end_auction

Recommendation: Please update the balance to zero before doing the transfer.

Status: Fixed

2. Using comparison instead of assignment

In several places in the code, there is an equality comparison operator (==) used instead of the value assignment (=). The compiler will not raise any errors because that's a totally valid construction the only warning could be that the result is never used. But from the sight of the logic, that's totally incorrect.

As an example let's see on this statement:

```
tOnSale[tId] == false; // When token is being purchased or
auctioned, by default it will be reset to NOT For SALE
```

While the comment says it is "reset to NOT For SALE", actually nothing happens here. Just because it's a comparison, not an assignment.

Contracts: Colexion

Functions: _tTransfer, enableTokenSale, mint

Recommendation: Please correct comparison and assignment operators usages.

Status: Fixed

High

1. Inconsistency with fees

While in the comments it says: "transfer 7% to token manager; transfer 10% to token creator; transfer 83% to token owner" it is actually doesn't do what it said.

The function, actually, accepts the “percents” values as its arguments and those arguments are never checked.

Using this function anyone buying a token could specify any percents they want.

For example, anyone could call the function with m=0, r=0, o=0; in this case, everyone: manager, creator and owner will receive 0 amount and the attacker will receive this token for free.

Contracts: Colexion

Function: purchaseToken

Recommendation: Please make sure that fees are checked correctly.

Status: Fixed

2. No way to put a token on “not for sale”

Regarding the documentation it says:

- The creator or owner of the token can place it on buy, auction or not on sale.
- When the user buy the NFT it is by default on not for sale

But in the code there is no way to put the item on “not for sale”. Even more, there is no way to disallow to directly buy the item while it’s on auction.

So, each token could be bought no matter if the owner/creator wants not to sell or even wants to make an auction.

Contracts: Colexion

Function: purchaseToken

Recommendation: Please make sure that the token could be “not for sale” or “auction only”.

Status: Fixed

Medium

No medium severity issues were found.

Low

1. No event on token price updated

When updating the price of the token it is a good idea to emit an event. This will make it easier to track such changes off-chain

Contracts: Colexion

Function: updateTokenPrice

Recommendation: Please emit an event on token price changes.

Status: Fixed

2. No event on the auction bid

When bid is placed it is a good idea to emit an event. This will make it easier to track such changes off-chain

Contracts: Colexion

Function: bid

Recommendation: Please emit an event on auction bid.

Status: Fixed

3. No event on auction created

When an auction is created for a token it is a good idea to emit an event. This will make it easier to track such changes off-chain

Contracts: Colexion

Function: create_Auction

Recommendation: Please emit an event on an auction is created.

Status: Fixed

4. No event on auction ended

When an auction is ended for a token it is a good idea to emit an event. This will make it easier to track such changes off-chain

Contracts: Colexion

Function: end_auction

Recommendation: Please emit an event on an auction is ended.

Status: Fixed

5. “token_bidders” is updated but never used

While there is an array that is always updated in the case of the bidder didn’t hit the maximum bid, but it is never used in the code, it’s just burning the gas while doing nothing.

Contracts: Colexion

Function: bid (lines: 552-554)

Recommendation: Please remove the “token_bidders” variable.

Status: Fixed

6. "bidders" field is updated but never used

While there is an array in the structure "Auction_data" that is always updated in the case of the bidder didn't hit the maximum bid, but it is never used in the code, it's just burning the gas while doing nothing.

Contracts: Colexion

Function: bid (lines: 557)

Recommendation: Please remove the "bidders" field from the structure.

Status: Fixed

7. Conformance to Solidity naming conventions

Solidity defines a naming convention that should be followed.

Rule exceptions:

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

Contracts: Colexion

Functions: t_transfer, create_Auction, end_auction,

Struct: Auction_data

Variables: T_In_Auction, T_Auction, token_bidders, bidder_list, T_cost

Modifiers: in_auction, auction_ended, only_owner, only_bidders, not_auction_winner

Recommendation: Follow the Solidity naming convention.

Status: Fixed

8. Boolean equality

Boolean constants can be used directly and do not need to be compared to **true** or **false**.

Contracts: Colexion

Functions: mint, purchaseToken, createAuction,

Modifiers: inAuction, notOnSale, auctionEnded

Recommendation: Remove equality to the boolean constant.

Status: Fixed

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

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **1** critical, **2** high and **7** low severity issues.

After the second review security engineers found **1** high and **3** low severity issues.

After the third review security engineers found that all previously found issues were resolved, but new ones were discovered. Therefore there are **1** high and **1** low severity issue.

After the fourth review security engineers found that **all issues** were addressed.

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 in accordance with the best industry practices at the date of this report, in relation to 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 audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. 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.

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 audit can't guarantee the explicit security of the audited smart contracts.