HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: CRE
Date: 23 May, 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 CRE
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.creproject.com/
Changelog	12.05.2023 - Initial Review 23.05.2023 - Second 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 CRE (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

CryptoRealEstate is an ERC-20 token that mints all initial supplies to a given address provided during contract creation. Additional minting is not allowed. It has the following attributes:

- Name: provided by deployer
- Symbol: provided by deployer
- Decimals: 18
- Total supply: 100m tokens.

CryptoRealEstate uses CustomOwnable contract to restrict access to critical functionalities, but `transferOwnership()` is not allowed. During contract creation, there is a new pair on PancakeSwap created - CRE/wETH.
Additionally, each transfer of these tokens is taxed with up to 9 % and tax is burned. Owner of the contract can add an address to the whitelist and transferring tokens to or from the whitelisted address omits tax. There are 3 available taxes, but only one is appalled per transfer:

- buyTax - up to 3%, this tax is applied when transfer is done from DEX.
- sellTax - up to 9%, this tax is applied when transferring to DEX. Half of the tax is burned and the second half is saved for marketing wallet.
- transferTax - up to 3%, this tax is applied when transferring to or from other addresses.

Privileged roles

- The owner of the *CryptoRealEstate* contract can or can remove a given address from the whitelist, change buy, sell and transfer tax, set marketing wallet address, open trading, withdraw ERC20 or native tokens from 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 **10** out of **10**.

- Functional requirements are detailed.
- A detailed technical description is provided in the form of extensive NatSpec comments.

Code quality

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

- Development environment is configured.
- Code follows best practices.

Test coverage

Code coverage of the project is **0%** (branch coverage).

- No tests.

Security score

As a result of the audit, the code contains no issues. 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: **10**.

The system users should acknowledge all the risks summed up in the risks section of the report.

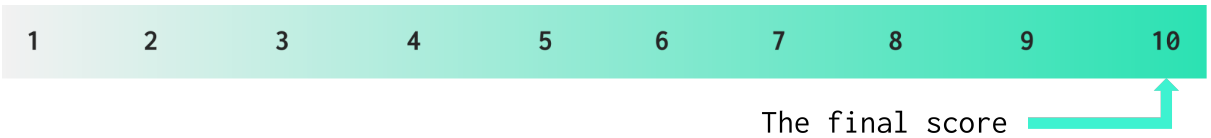

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 May 2023	7	1	0	0
23 May 2023	0	0	0	0

Risks

- If a Marketing Wallet address is a contract without the ability to withdraw native tokens, funds can be potentially locked.

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.	Passed	
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.	Passed	
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.	Passed	
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	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

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.	Not Relevant	
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.	Not Relevant	
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	
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.	Passed	
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.	Passed	
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.	Failed	
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

No critical severity issues were found.

High

No high severity issues were found.

Medium

M01. Requirements Violation

Impact	Low
Likelihood	High

In the *changeSwapAmount()* function, the comparison of the *_newThreshold* variable is done incorrectly.

It is stated in the revert message of the required function that its value should be: *0.005-0.03% of total supply*

The total supply of the CRE token is constantly decreasing due to the burning of collected fees, but the constant variable of the initial mint is used for comparison in the changeSwapAmount() function.

Path:
./contracts/CryptoRealEstate.sol : changeSwapAmount()

Recommendation: Use the current total supply in the changeSwapAmount() function when setting the new threshold.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3) (Requirements updated to *0.005-0.03% of initial total supply*.)

Low

L01. Unscalable Functionality - Copy Of Well-Known Contract

Impact	Low
Likelihood	Medium

It is considered that smart contract systems should be easily scalable.

Well-known contracts from projects like OpenZeppelin should be imported directly from source as the projects are in development and may update the contracts in future.

This may lead to new issues during further development and unexpected errors in case of accidental or inattentive modification.

Path:
./contracts/CryptoRealEstate.sol : *,

Recommendation: Import the contract directly from source. Override *transferOwnership()* and put *revert()* inside the overridden function body.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

L02. Unchecked Transfer

Impact	Low
Likelihood	Low

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The function does not use SafeERC20 library for checking the result of ERC20 token transfer. Tokens may not follow ERC20 standard and return false in case of transfer failure or not returning any value at all.

This may lead to denial of service vulnerabilities during interactions with non-standard tokens.

Path:
./contracts/CryptoRealEstate.sol : withdrawToken()

Recommendation: Follow common best practices, use SafeERC20 library to interact with tokens safely.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

L03. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Path:
./contracts/CryptoRealEstate.sol : constructor(), setTreasury()

Recommendation: Implement zero address checks.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

L04. Missing Events

Impact	Low
Likelihood	Low

Events for critical state changes should be emitted for tracking things off-chain.

Path:
./contracts/CryptoRealEstate.sol : constructor(), changeSwapAmount(), addToWhitelist(), removeFromWhitelist(), changeBuyTaxFeePercent(), changeTransferTaxFeePercent(), changeSellTaxFeePercent(), setTreasury(), openTrading(), withdrawToken(), withdrawBNB()

Recommendation: Create and emit related events.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

L05. CEI Pattern Violation

Impact	Medium
Likelihood	Low

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The Checks-Effects-Interactions pattern is violated. During the function, checks are done after state variables are updated.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

Path:
./contracts/CryptoRealEstate.sol : constructor()

Recommendation: Follow common best practices, and implement the function according to the Checks-Effects-Interactions pattern.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

L06. Contradiction - Undocumented Functionality

Impact	Medium
Likelihood	Low

It is considered that the project should be consistent and contain no self-contradictions.

The *changeSwapAmount()* is never documented. Range that the new threshold must be is undocumented.

This may lead to unexpected contract behavior.

Path:
./contracts/CryptoRealEstate.sol : changeSwapAmount()

Recommendation: Provide documentation, comments and identifiers in code consciously, remove the functionality or mention it in documentation.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

L07. Contradiction - Documentation Mismatch

Impact	Medium
Likelihood	Low

It is considered that the project should be consistent and contain no self-contradictions.

According to documentation, the *withdrawToken()* function allows the contract owner to withdraw any BSC token accidentally sent to the contract. However, in the implementation, the owner can also withdraw funds for marketing wallet before the balance of contract is greater than threshold.

This may lead to unexpected contract behavior.

Path:
./contracts/CryptoRealEstate.sol : withdrawToken()

Recommendation: Fix the mismatch.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

Informational

I01. State Variables That Can Be Packed

Since the state variables in the *CryptoRealEstate* contract *sellTaxFeePercent*, *transferTaxFeePercent* and *buyTaxFeePercent*, represent numbers in range [0-9], they can be downcast and packed together in order to save Gas.

Paths:
./contracts/CryptoRealEstate.sol : sellTaxFeePercent, transferTaxFeePercent, buyTaxFeePercent

Recommendation: Consider downcasting the mentioned variables to smaller uint sizes and place them next to each other in order to pack storage.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

I02. Floating Pragma

Locking the pragma helps 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/CryptoRealEstate.sol : *
./contracts/IPancakeFactory.sol : *
./contracts/IPancakeRouter.sol : *
./contracts/CustomOwnable.sol : *

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

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

I03. Style Guide Violation - Order of Functions

The project should follow the official code style guidelines.

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

Within a grouping, place the `view` and `pure` functions at the end.

Paths:
./contracts/CryptoRealEstate.sol
./contracts/CustomOwnable.sol

Recommendation: The official Solidity style guidelines should be followed.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

I04. Unused Functions

The functions created but not used in the project should be deleted. This will make a more readable contract.

Path:
./contracts/IPancakeRouter.sol : swapExactETHForTokensSupportingFeeOnTransferTokens(), swapExactTokensForETH(), addLiquidityETH()

Recommendation: Remove unused function.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

I05. Redundant Code Block

According to documentation, `withdrawBNB()` only purpose is to withdraw native tokens stuck in contract.

Only function that accepts native tokens to contract is *`receive().`*

This function does not have a body and is not needed to make the contract work properly.

Path:
./contracts/CryptoRealEstate.sol : withdrawBNB(), receive()

Recommendation: Delete redundant code block.

Found in: 960021a

Status: Reported

I06. Missing Empty String Check

Missing empty string check for *`_name`* and *`_symbol`* of newly created ERC20 token. These values cannot be changed later.

Path:
./contracts/CryptoRealEstate.sol : constructor()

Recommendation: Implement empty string checks.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

I07. Typo

The pancekeRouter state variable has a typo in its name.

Path:
./contracts/CryptoRealEstate.sol : pancekeRouter

Recommendation: Fix the typo.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

I08. Style Guide Violation - Naming Conventions

Constants should be named with all capital letters with underscores separating words.

For example: TOTAL_SUPPLY.

Path:
./contracts/CryptoRealEstate.sol : TOTALSUPPLY

Recommendation: Fix the violation.

Found in: 960021a

Status: Fixed (Revised commit: 2de82e3)

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.

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

Appendix 2. Scope

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

Initial review scope

Repository	https://dev.azure.com/blockstars/Solidity_Projects_Public/_git/CREToken
Commit	960021aa8ac1e55820688d7ceedbc405453403cc
Whitepaper	https://www.creproject.com/whitepaper
Requirements	CRE Token.docx SHA3: 2620f0d50278aab1d5185bd6e12fef755940a8b04e4a23cf13f7d12d070e79c9
Technical Requirements	CRE Token.docx SHA3: 2620f0d50278aab1d5185bd6e12fef755940a8b04e4a23cf13f7d12d070e79c9
Contracts	File: contracts/CryptoRealEstate.sol SHA3: 54aa0c5bd4d60c4384ea78273fc288e7c42f47a710b57604e7cb3c2c89fe6042 File: contracts/CustomOwnable.sol SHA3: e33f9c489af7e4a864f208454df4809435b0eb5d3bee582f21ce0384d1c7152f File: contracts/interfaces/IPancakeFactory.sol SHA3: d5043b2256e646a0b999e2b5f20a6a75d8885cbf988b1b4117e39fde0f9d2352 File: contracts/interfaces/IPancakeRouter.sol SHA3: d6d27eeb75bd1879e9782c9b3595eac2e0628de3445e32c01eb2efd232873552

Second review scope

Repository	https://dev.azure.com/blockstars/Solidity_Projects_Public/_git/CREToken
Commit	2de82e38f61602c2c79e85627f415740ef8b7ddf
Whitepaper	https://www.creproject.com/whitepaper
Requirements	CRE Token.docx SHA3: 2620f0d50278aab1d5185bd6e12fef755940a8b04e4a23cf13f7d12d070e79c9 CRE Token Requirements.pdf SHA3: ceee09fd9b8f2b9c9aff06d06f200278059d0f560d4a0f0d331421f41db852b3
Technical Requirements	CRE Token.docx SHA3: 2620f0d50278aab1d5185bd6e12fef755940a8b04e4a23cf13f7d12d070e79c9 CRE Token Requirements.pdf SHA3: ceee09fd9b8f2b9c9aff06d06f200278059d0f560d4a0f0d331421f41db852b3
Contracts	File: contracts/CryptoRealEstate.sol SHA3: 194b7f5cfd9c3b4fbdd2dab0cd5989698fcc2d5a7783971c84c530738d1960d1 File: contracts/interfaces/IPancakeFactory.sol SHA3: 1666978c0da6805296cddd6d706e4785ed5ef7ec8c356c2a2ea21768d0043e8b File: contracts/interfaces/IPancakeRouter.sol SHA3: 373f240a2b4af2895ac3248670c1477a0fe27c30fbd65824c2d0a69822cf10e8

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