HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: CryptoCookies
Date: June 28th, 2022

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 CryptoCookies
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Betting application
Platform	EVM
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://www.cryptocookies.com/
Timeline	17.06.2022 - 28.06.2022
Changelog	22.06.2022 - Initial Review 28.06.2022 - 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 CryptoCookies (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.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Code archive:
cryptocookies-423695870-main.zip
Archive hash:
c5045eab818fe19e44fd09960ba187bd881ba248837800ad73f2f719e2071d7d
Technical Documentation:
Type: Readme (Brief project overview)
Link
Integration and Unit Tests: Yes
Contracts:
File: ./cryptocookies-app/contracts/src/main/solidity/CookieHolder.sol
SHA3: 76e87acc7bd19a389f3b317227037e70549d6ee7fa3cca2cd2b1d694d26ba063

File: ./cryptocookies-app/contracts/src/main/solidity/TestDependencies.sol
SHA3: 0493cb318b2f891fa822d66510b0e8f2b74ba4674db9906ecaeab6878f25a4c4

Second review scope

Repository:
https://github.com/cryptocookies-dev/CryptoCookiesContracts
Commit:
fee464dedbdf2bb1b68df8e182930b5abcbc6f79
Technical Documentation:
Type: Readme (Brief project overview)
Link
Integration and Unit Tests: Yes
Contracts:
File: ./src/main/solidity/CookieHolder.sol
SHA3: 8d19ec60b1fb14cda869c5b9e7d2f8a51e491739042f7616a1fef0ae840b7c08

File: ./src/main/solidity/TestDependencies.sol
SHA3: c28f4eca023e4d60236ae51a08001b9ab4cdc9a83226f5666ac24cb243ab529b

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 cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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 methodology.

Documentation quality

The total Documentation Quality score is **5** out of **10**. A brief project overview is provided in the Readme file in the repository. Functional and technical requirements were not provided. Code is followed by NatSpec comments.

Code quality

The total CodeQuality score is **10** out of **10**. Code follows Style guide. Code is covered with tests.

Architecture quality

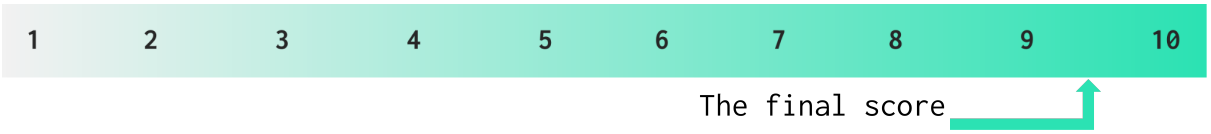
The architecture quality score is **10** out of **10**. Contract uses best practices.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.5**.



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	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	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	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	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	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.	Passed
Token Supply manipulation	**Custom**	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	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution	Passed

		fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

CryptoCookies is an application for betting on crypto rate:

- *CookieHolder* – contains functionality for registering tokens, creating deals, withdrawing rewards, and working with the deposits. This contract receives, holds, and transfers tokens to the destination address.
- *TestDependencies* – test contract for implementing the demo app.

Privileged roles

- DEFAULT_ADMIN_ROLE: Admin role, can re-register token, pause and unpause token
- DEALER_ROLE: can register token, confirm deal, settle deals, reject deals, close deals and reject deal close
- TREASURY_ROLE can withdraw settlement, recalculate settlement balances, and deposit settlement

Risks

- Part of the project`s logic is implemented **off-chain**, so we may not guarantee the secureness of the logic parts **not** implemented in the contract. Backend off-chain logic implements rewards calculation logic, transfers calling logic, and treasury management.
- In case of miscalculations in the backend treasury system and insufficient tokens funding, contract treasury can become empty, and users will not be able to withdraw their rewards.
- The destination address in transfer functions is set in the function parameters and called by a backend. In case of backend error, the destination address can be set with a mistake, and funds will be lost.

Findings

Critical

No critical severity issues were found.

High

1. **Undocumented pausing functionality.**

   The functionality allows the owner to pause all the token transfers anytime. Pausing functionality should be limited by clear contract rules. The documentation does not mention the functionality of transfers stopping.

   This can lead to users' funds manipulation.

   Contract: CookieHolder

   Function: pause

   Recommendation: Remove pausing functionality or notify users about it in the provided documentation.

   Status: Fixed (revised commit: fee464d)

Medium

1. **Costly operations inside a loop.**

   The loops inside contract`s functions are not optimized. Loops read the state and change state variables. Making this computation over and over again will be costly in terms of Gas.

   This can lead to high Gas consumption.

   Contract: CookieHolder

   Functions: initialize, registerToken, recalcSettlementBalances, getBalances, settleDetails, _assertDealStatusIn

   Recommendation: Cache the array length, save state variables to local memory, iterate the loop and then save the values to the state.

   Status: Fixed (revised commit: fee464d)

2. **Contract code size exceeds 24576 bytes.**

   This contract may not be deployable on mainnet.

   Contract: CookieHolder

   Functions: -

   Recommendation: Use the optimizer.

   Status: Fixed (revised commit: fee464d)

Low

1. **State variables default visibility.**

   Variable`s *tokens, tokenSymbols, settlementBalances, openDealBalances, tokenContracts, totalPayoutAtZero, totalPayoutAtInf, MIN_BALANCE_MULTIPLIER* visibility is not specified. Specifying state variables visibility helps to catch incorrect assumptions about who can access the variable.

   This makes the contract`s code quality and readability higher.

   Contract: CookieHolder

   Functions: -

   Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

   Status: Fixed (revised commit: fee464d)

2. **Using SafeMath in Solidity ^0.8.0.**

   Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.

   There is no need to use the SafeMath library.

   Contract: CookieHolder

   Functions: -

   Recommendation: Remove the SafeMath library.

   Status: Fixed (revised commit: fee464d)

3. **Some functions can be declared external.**

   Public functions that are never called in a contract should be declared external.

   Contract: CookieHolder

   Functions: initialize, reregisterToken

   Recommendation: Declare mentioned functions as external.

   Status: Fixed (revised commit: fee464d)

4. **Redundant functions.**

   Contract has a receive function, which is reverting any calls. Such reverting is a standard contract behavior, which makes these function redundant.

   Contract: CookieHolder

   Functions: receive

   Recommendation: Remove redundant functions.

Status: Fixed (revised commit: fee464d)

5. **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.

   Contract: CookieHolder, TestDependencies

   Functions: -

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

   Status: Fixed (revised commit: fee464d)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 cannot guarantee the explicit security of the audited smart contracts.