HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Lunabets
Date: September 9, 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 Lunabets
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Gambling
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://liongaming.io/
Timeline	11.07.2022 - 05.09.2022
Changelog	20.07.2022 - Initial Review 11.08.2022 - Second Review 09.09.2022 - Third Review

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

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Lunabets (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

Repository:
https://github.com/casinobitcoin/lunafi-smart-contracts
Commit:
fd30241

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/LionGaming.sol
SHA3: a94002421d63c5ef09ba9b07c4d5ac708ff6da3641d81a0efd3385f817ec2d95

File: ./contracts/Sportsbook.sol
SHA3: 8092d259e769ee6b6c40b8c1dfc7a139f999b00356133234fdcd1830df9cbcff

Second review scope

Repository:
https://github.com/casinobitcoin/lunafi-smart-contracts
Commit:
64abd35

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/LionGaming.sol
SHA3: 8a65bfa33ee9ffbffc0e63debea92858714536975c95bc3df8ac7660670c24f6

File: ./contracts/Sportsbook.sol
SHA3: a6858deb5dd9775694081dd1c3efbfbdf5e49d3b24352dba3c75e4d5975c4415

Third review scope

Repository:
https://github.com/casinobitcoin/lunafi-smart-contracts
Commit:
4f3c77d

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/LionGaming.sol
SHA3: 62ac5726ff02575b848412e13cde27772592d1db83dc16e67e96181a57060f2d

File: ./contracts/Sportsbook.sol
SHA3: a6858deb5dd9775694081dd1c3efbfbdf5e49d3b24352dba3c75e4d5975c4415

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 **1** out of **10**. Functional requirements are not provided. A technical description is not provided. NatSpec comments are partially missing and contain abbreviations and typos.

Code quality

The total CodeQuality score is **5** out of **10**. Code follows Style guide. Unit tests were provided. **Solidity coverage fails due to repository inconsistency.**

Architecture quality

The architecture quality score is **10** out of **10**.

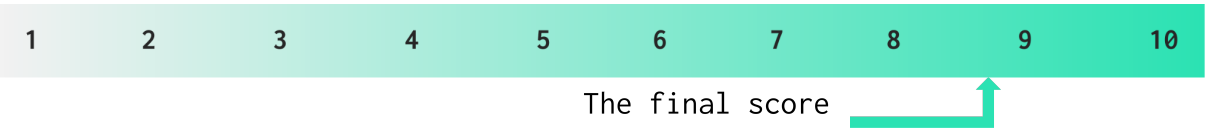
Security score

As a result of the audit, the code contains **2** low severity 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: **8.6**.



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

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.	Not Relevant
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.	Not Relevant
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	Not Relevant

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. All parameters from the signature should be used in signer recovery	Passed
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.	Passed
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.	Not Relevant
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	Passed

		data stored on the contract. There should not be any cases when execution 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
Secure Oracles Usage	**Custom**	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	**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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Failed

System Overview

LionGaming is a gambling platform with the following smart contracts:

- *LionGaming.sol* - a contract that manages user funds on the platform.
- *Sportsbook.sol* - a smart contract where agents can place bets on behalf of the users.

Privileged roles

- ADMIN_ROLE can enable deposits, add coins, update HousePool addresses, set LFI token index in *LionGaming.sol* and set LionGaming contract in *Sportsbook.sol*
- AGENT_ROLE can evict balance, withdraw, withdraw to hosted, assign affiliates, distribute affiliate earnings, sweep hosted wallets and distribute LFI in *LionGaming.sol* and update EVME place bets, place parlay bets, settle bets, settle parlay legs, and update flags in SportsBook.sol
- COMMISSION_MGR_ROLE can withdraw commission in LionGaming.sol
- RATES_SETTER_ROLE can set Lion commission and set house edge discount in LionGaming.sol
- DEPOSIT_MGR_ROLE can call *internalDeposit* function in LionGaming.sol
- TREASURER_ROLE can fund contract, fund contract tokens, and withdraw contract funds in LionGaming.sol
- CLIENT_CONTRACT_ROLE can transfer funds to pool, reduce balance and take commission, increase balance and return commission, increase balance, commit commission, giveLFI rewards, and sync balance in LionGaming.sol

Risks

- In case of an admin keys leak, an attacker can get access to funds that belong to users.

Findings

Critical

1. **Compilation issues**

   File imports IERC20WithMetaTx, DateTime, HousePool(LionGaming), HousePool(Sportsbook) are invalid and not referencing required files.

   The contract cannot be compiled.

   Contracts: Sportsbook, LionGaming

   Recommendation: Check file imports.

   Status: Fixed (64abd35)

High

1. **Highly permissive role access**

   Manager can change *lionGaming, housePool, lfiCoinIndex*

   This can lead to user funds manipulations.

   Contracts: Sportsbook, LionGaming

   Functions: setLionGamingContract, updateHousePoolAddress, setLFICoinIndex

   Recommendation: Add highly permissive functionality to documentation.

   Status: Fixed (64abd35)

Medium

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

   Starting with Solidity ^0.8.0, SafeMath functions are built-in. Due to this, using this library is redundant.

   Contracts: Sportsbook, LionGaming

   Recommendation: Remove redundant functionality.

   Status: Fixed (64abd35)

2. **Concrete implementation instead of an interface**

   Importing interfaces instead of implementations allows to save more Gas and reduces the contract size.

   Contract: LionGaming

   Variable: lionGaming

   Recommendation: Create ILionGaming interface and change the data type to ILionGaming.

   Status: Fixed (64abd35)

3. **Redundant logic**

   Logical checks that duplicate the logic of the previous statements or do not perform any validation can be removed to save Gas.

   Contracts: Sportsbook, LionGaming

   Functions: initalize:122(Sportsbook), initalize:140(LionGaming), addCoin:182

   Recommendation: Remove redundant logic.

   Status: Fixed (64abd35)

4. **Insufficient check**

   The insufficient check may result in function execution with empty data.

   This spends Gas.

   Contracts: Sportsbook, LionGaming

   Functions: settleBets, commitCommission

   Recommendation: Check if the token address is valid.

   Status: Fixed (64abd35)

Low

1. **Redundant pragma statement**

   Pragma ABIEncoderV2 will be activated by default starting from Solidity 0.8.0.

   Contract: Sportsbook

   Recommendation: Remove redundant pragma.

   Status: Fixed (64abd35)

2. **Tight variable packing**

   Rearranging variables into single memory slots allows to save more Gas.

   Contracts: Sportsbook, LionGaming

   Structs: ParlayBetSlip, UserPermission, SweepRequest

   Recommendation: Cache array length.

   Status: Reported

3. **State variables' default visibility**

   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: LionGaming

Variable: lionGaming

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

Status: New

4. **Unused state variables**

   Some state variables are declared but never used.

   Contract: LionGaming

   Variable: creationBlock

   Recommendation: Remove unused variables.

   Status: Fixed (64abd35)

5. **Costly type conversion**

   Converting *address* argument to *IERC20WithMetaTx* or *HousePool* interface type is a costly operation that consumes a lot of Gas.

   Contract: LionGaming

   Functions: addCoin, updateHousePoolAddress, fundContractTokens, makePayment, sweepHostedWallets

   Recommendation: Pass arguments as interface types.

   Status: Fixed (64abd35)

6. **Code optimization possibility**

   Usage of different advanced programming techniques allows to save some Gas.

   Contracts: Sportsbook, LionGaming

   Functions: placeBets, placeParlayBets, placeBet, settleBets, updateBetFlags, hashStraightBets, calculateParlayOdds, assignAffiliates, distributeAffiliateEarnings, sweepHostedWallets, distributeLFI (:422)

   Recommendation: Cache array length, optimize loop by removing require, cache state variables.

   Status: Fixed (64abd35)

7. **Check-Effect-Interaction pattern violation**

   Function firstly makes interaction with other function, and only after that changes state variable.

   Contract: LionGaming

   Functions: withdrawCommission, withdrawCommissionTo

Recommendation: Change state variable before making a function call.

Status: Fixed (64abd35)

8. **Code repetition**

   Code contains repetitive *require* (coin index check, supported tokens check, deposits enabled check) that make the smart contract harder to read.

   Contracts: Sportsbook, LionGaming

   Functions: evictBalance, withdraw, withdrawToHosted, internalDeposit, fundContractTokens, withdrawContractFunds, withdrawContractFundsTo, reduceBalanceAndTakeCommission, increaseBalanceAndReturnCommission, increaseBalance, getBalance, getAffiliateBalance, _deposit, _depositToken.

   Recommendation: Use function modifiers.

   Status: Fixed (64abd35)

9. **Redundant validations**

   The function flow contains requires that can be simplified.

   No need to double check coinIndex.

   Contract: LionGaming

   Functions: _depositToken:782, :788

   Recommendation: Check the function flow.

   Status: Fixed (64abd35)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.