HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Shark Race
Date: March 18th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Shark Race.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	BEP20 token
Platform	Binance Smart Chain / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://gitlab.com/transcrypt/sharkrace/smartcontracts
Commit	465d19e4c8f155d924fa3ecc3b3477acb5a901ff
Technical Documentation	NO
JS tests	YES
Website	https://sharkrace.com
Timeline	01 FEBRUARY 2022 - 18 MARCH 2022
Changelog	10 FEBRUARY 2022 - Initial Audit 24 FEBRUARY 2022 - Second Review 18 MARCH 2022 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Shark Race (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 February 1st, 2022 - February 10th, 2022.

The second review was conducted on February 24th, 2022.

The third review was conducted on March 18th, 2022.

Scope

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

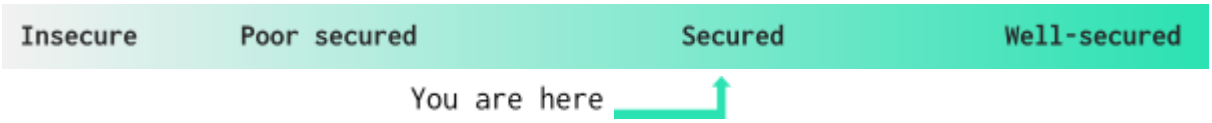
Repository:
https://gitlab.com/transcrypt/sharkrace/smartcontracts
Commit:
465d19e4c8f155d924fa3ecc3b3477acb5a901ff
Technical Documentation: No
JS tests: Yes – included in “test” directory
Contracts:
interfaces/IHook.sol
hooks/HookBlackWhiteList.sol
lib/ERC20Hooks.sol
lib/ERC20Vesting.sol
lib/errors.sol
test/CoinMock.sol
test/Hook2Mock.sol
test/Hook1Mock.sol
Coin.sol

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 secured but not fully match the given documentation.



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 **2** critical, **4** high, **1** medium, and **4** low severity issues.

After the second review security engineers found that most contracts were rewritten, new contracts were added to the scope, and some were removed. However, there are still **2** medium and **1** low severity issues.

After the third review security engineers found that the previous documentation provided was discarded totally like most contracts. The `antisnipe` functionality was fully removed as well as the `fees` one. The `BEP721 token` and `BEP721 Seller` functionality was fully removed from the scope. Also, transferring to and the vesting is now allowed for the token contract itself. Therefore, **no security issues** were found.

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

Audit overview

■■■■ Critical

1. Tokens locked forever.

The *ERC20Funds._proceedHookLP* function does nothing and is not overridden, so "_amountToFunds" function will always subtract "_amountToLP" from the contract's balance and the corresponding balance would be locked forever.

Contracts: ERC20Funds.sol

Function: _proceedHookLP

Recommendation: Please consider implementing "*_proceedHookLP*" function.

Status: Functionality removed.

2. Not possible to sell not all cards.

The *CardSeller.startSale* function requires the seller to sell all Cards seller has, disregarding types and the willingness to sell part of cards.

Contracts: CardSeller.sol

Function: startSale

Recommendation: Please revise the selling process. Maybe you'd better look to the EIP 1155 instead of 721.

Status: Functionality removed.

■■■ High

1. Incorrect mapping changed.

When calling the *SharkCoin.setForcedFee* function, it changes the "_isExcludedFee" mapping instead of the "_isForcedFee" one.

Contracts: SharkCoin.sol

Function: setForcedFee

Recommendation: Fix the assignment in the function above.

Status: Functionality removed.

2. No pool transfers are checked.

While it stated clearly in the docs:

- setup buying fees (transfers from the pool)
- setup selling fees (transfers to the pool)

the contract never checks for the “to” or “from” to be the pool address. But it checks the "_isForcedFee” mapping before applying the buy or sell fees, which is kinda strange behavior.

Contracts: SharkCoin.sol

Function: _calcFee

Recommendation: Please revise fees collection.

Status: Functionality removed.

3. Fees could not be sent to the address.

While it stated clearly in the docs:

- collected fees could be periodically burned and/or sent to the address (ratio to be set)

the contract never tries to send collected fees to some address. But there are “_pcntLP” and “_pcntBurn” that look similar to the requested, but fees collected by the “_pcntLP” coefficient are never being sent anywhere.

Contracts: SharkCoin.sol

Function: _proceedFunds

Recommendation: Please revise fees sendings.

Status: Functionality removed.

4. Incorrect NFT standard.

Looking through the Card contract and how it’s being used in the CardSeller contract, we’re thinking that the chosen 721 eip doesn’t suit for needs. Looks like the “EIP-1155: Multi Token Standard” could be more helpful.

Contracts: Card.sol

Recommendation: Please consider checking the Card NFT implementation.

Status: Functionality removed.

Medium

1. Low test coverage.

Overall test coverage is low. It’s about 74% for statements and only 51% for code branches.

Scope: tests

Recommendation: Increase code coverage to at least 95% for statements and 100% for code branches.

Status: Fixed.

2. Fees not taken.

While it is stated clearly in the docs, that contract should take fees for transferring from/to pool, the contract never tries to collect any fees.

Contracts: Coin.sol

Function: _beforeTokenTransfer

Recommendation: Please revise fees collection or remove it from docs.

Status: Fees functionality removed.

3. Weak PRNG.

Weak PRNG due to a modulo on block.timestamp, now or blockhash. These can be influenced by miners to some extent so they should be avoided.

Contracts: ShuffleId.sol

Function: diceRoll

Recommendation: Do not use block.timestamp, now, or blockhash as a source of randomness. Please consider using off-chain-VRFs, like Chainlink or others for real randomness and not relying on malicious miners.

Status: Functionality removed.

Low

1. Reading the state in the loop.

The for-loop reads the "_funds" state variable in the loop which spends gas.

Contracts: ERC20Funds.sol

Function: _splitToFunds

Recommendation: Please read the state value into the local memory variable and then iterate.

Status: Functionality removed.

2. Deprecated pragma.

Since solidity version 0.8.0 ABI coder v2 is activated by default. You can choose to use the old behaviour using "**pragma abicoder v1;"**. The pragma "**pragma experimental ABIEncoderV2;"** is still valid, but it is

deprecated and has **no** effect. If you want to be explicit, please use "**pragma abicoder v2;"** instead.

Contracts: SharkNFT.sol

Recommendation: Please either remove deprecated pragma or use explicit declaration if you want to (pragma abicoder v2;).

Status: Functionality removed.

3. Redundant expression.

Below expressions are referenced a function argument but perform no action with them, so no code will be generated for such statements and they can be removed.

Expressions:

- newImplementation (Card.sol#209)
- newImplementation (CardsSeller.sol#261)
- newImplementation (SharkCoin.sol#350)
- newImplementation (SharkNFT.sol#217)
- from (lib/ERC20Fee.sol)
- to (lib/ERC20Fee.sol)
- amount (lib/ERC20Fee.sol)
- amount (lib/ERC20Fee.sol)
- amount (lib/ERC20Funds.sol)
- amount (lib/ERC20Funds.sol)

Recommendation: Remove redundant expressions.

Status: Fixed.

4. A public function that could be declared external

public functions that are never called by the contract should be declared **external**.

Contracts: Card.sol, CardsSeller.sol, SharkCoin.sol, SharkNFT.sol, ERC20Fee.sol, ERC20Funds.sol, HasTokenURI.sol

Functions: Card.initialize, Card.setBaseURI, Card.setTemplateURI, Card.mint, Card.mintBatch, Card.burn, Card.pause, Card.unpause, Card.use, Card.useFrom, CardSeller.initialize, CardSeller.getSaleRemainAmount, CardSeller.getMintPassCardSaleRemainAmount, CardSeller.getSalePrice, CardSeller.getMintPassCardSalePrice, CardSeller.buy, SharkCoin.initialize, SharkCoin.isExcludedFee, SharkCoin.isForcedFee, SharkCoin.setFees, SharkCoin.setFundPercnents, SharkCoin.setFunds, SharkCoin.pause, SharkCoin.unpause, SharkNFT.initialize, SharkNFT.setBaseURI, SharkNFT.setTemplateURI, SharkNFT.mint, SharkNFT.burn, SharkNFT.lock, SharkNFT.unlock, SharkNFT.lockedBy, SharkNFT.pause, SharkNFT.unpause, ERC20Fee.name, ERC20Fee.symbol,

ERC20Fee.decimals, ERC20Fee.totalSupply, ERC20Fee.transfer, ERC20Fee.approve, ERC20Fee.transferFrom, ERC20Fee.increaseAllowance, ERC20Fee.decreaseAllowance, ERC20Fee.burn, ERC20Fee.burnFrom, ERC20Funds.getFundPercents, ERC20Funds.getFunds, ERC20Funds.getFundAmounts, HasTokenURI.baseURI, HasTokenURI.templateURI

Recommendation: Use the **external** attribute for functions never called from the contract.

Status: Fixed.

5. Events not emitted.

It is recommended to contract function to emit an event when crucial state variable is changed. That gives more clarity and an ability for the community to track such changes off-chain.

Contracts: CardSeller.sol, NFTSeller.sol

Functions: CardSeller.setWallet, CardSeller.setPriceFeed, CardSeller.setTokenPair, CardSeller.setCardContract, CardSeller.setActiveSaleCardType, CardSeller.openSale, CardSeller.closeSale, CardSeller.buy, NFTSeller.setWallet, NFTSeller.setPriceFeed, NFTSeller.setTokenPair, NFTSeller.setCardContract, NFTSeller.openSale, NFTSeller.closeSale, NFTSeller.setSaleCardParams, NFTSeller.buyWithCard, NFTSeller.buyWithCardForTokenBUSD, NFTSeller.buyWithCardForToken

Recommendation: Please emit events on crucial state changes.

Status: Functionality removed.

6. Overriding function is missing "override" specifier.

Functions that override virtual functions or interface functions should have an “override” modifier in the declaration.

Contracts: Artifact.sol

Functions: Artifact.name, symbol, mint, mintBatch, burn, burnBatch

Recommendation: Please add “override” attribute to overridden functions.

Status: Functionality removed.

7. Invalid contract specified in override list.

The function overriding list includes contract (interface) which is not listed in the “is” section of the contract.

Contracts: Card.sol

Functions: isApprovedForAll

Recommendation: Please emit events on crucial state changes.

Status: Functionality removed.

8. Costly operations in the loop.

Incrementing the state variable in the loop is the anti-pattern which only burns gas.

Contracts: Card.sol

Functions: mintBatch

Recommendation: Please use the local memory variable to increment the “tokenIdTracker” and write it after the loop.

Status: Functionality removed.

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 **2** critical, **4** high, **1** medium, and **4** low severity issues.

After the second review security engineers found that most contracts were rewritten, new contracts were added to the scope, and some were removed. However, there are still **2** medium and **1** low severity issues.

After the third review security engineers found that the previous documentation provided was discarded totally like most contracts. The `antisnipe` functionality was fully removed as well as the `fees` one. The `BEP721 token` and `BEP721 Seller` functionality was fully removed from the scope. Also, transferring to and the vesting is now allowed for the token contract itself. Therefore, **no security issues** were found.

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.