SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Leonicornswap
Date: October 8th, 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 Leonicornswap.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	DEX Swap; Lottery; Farming
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Leonicornswap/DEXContracts
Commit	a68cb149cac50dadf11dc38db72e0f7cf24d3781.zip
Technical Documentation	NO
JS tests	YES
Timeline	27 SEPTEMBER 2021 - 04 OCTOBER 2021
Changelog	04 OCTOBER 2021 - Initial Audit 08 OCTOBER 2021 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Leonicornswap (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 September 27th, 2021 - October 4th, 2021.

Second review conducted on October 8th, 2021.

Scope

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

Repository:
https://github.com/Leonicornswap/DEXContracts
Commit:
a68cb149cac50dadf11dc38db72e0f7cf24d3781
Technical Documentation: No
JS tests: Yes
Contracts:

```
leonicorn-lottery:
	access\Ownable.sol
	chainlink\LinkTokenInterface.sol
	chainlink\VRFConsumerBase.sol
	chainlink\VRFRequestIDBase.sol
	interfaces\ILeonicornSwapLottery.sol
	interfaces\IRandomNumberGenerator.sol
	libraries\SafeMath.sol
	security\ReentrancyGuard.sol
	token\IERC20.sol
	token\SafeERC20.sol
	utils\Address.sol
	utils\Context.sol
	LeonicornLottery.sol
	RandomNumberGenerator.sol

leonicorn-farming:
	access\Ownable.sol
	interfaces\IBEP20.sol
	interfaces\IMasterHunter.sol
	interfaces\IMigratorHunter.sol
	libraries\SafeBEP20.sol
	math\Math.sol
	math\SafeMath.sol
	security\Pausable.sol
	security\ReentrancyGuard.sol
	token\ERC20\ERC20.sol
	token\ERC20\IERC20.sol
	token\ERC20\SafeERC20.sol
	utils\Address.sol
	utils\Context.sol
	BEP20.sol
	LeonToken.sol
	LeonVault.sol
	MasterHunter.sol
```

```
		RewardHolder.sol
	SmartHunterFactory.sol
	SmartHunterInitializable.sol

leonicorn-swap-core:
	GSN\Context.sol
	interfaces\ILeonicornCallee.sol
	interfaces\ILeonicornERC20.sol
	interfaces\ILeonicornFactoryV2.sol
	interfaces\ILeonicornPairV2.sol
	libraries\Math.sol
	libraries\UQ112x112.sol
	math\SafeMath.sol
	token\ERC20\ERC20.sol
	token\ERC20\IERC20.sol
	token\ERC20\SafeERC20.sol
	utils\Address.sol
	LeonicornERC20.sol
	LeonicornFactoryV2.sol
	LeonicornPairV2.sol

leonicorn-swap-periphery:
	access\Ownable.sol
	interfaces\ILeonicornFactoryV2.sol
	interfaces\ILeonicornPairV2.sol
	interfaces\ILeonicornRouter01.sol
	interfaces\ILeonicornRouter02.sol
	interfaces\IWBNB.sol
	libraries\LeonicornLibrary.sol
	libraries\TransferHelper.sol
	math\Math.sol
	math\SafeMath.sol
	token\ERC20\ERC20.sol
	token\ERC20\IERC20.sol
	token\ERC20\SafeERC20.sol
	utils\Address.sol
	utils\Context.sol
	LeonicornRouterV2.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 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 **3** high, **4** medium and **6** low severity issues.

After the second review security engineers found that **all** issues were addressed or commented on by the customer.

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

No critical issues were found.

🟨 🟨 🟨 High

1. Possible rewards lost or receive more

   Changing **allocPoint** in the MasterHunter.set method while **_withUpdate** flag set to **false** may lead to rewards lost or receiving rewards more than deserved.

   Recommendation: Please call updatePool(_pid) in the case if **_withUpdate** flag is **false** and you don't want to update all pools.

   Fixed before the second review

2. LeonToken leakage

   When staking, MasterHunter contract mints LeonTokens to the stakeholder. When leaving staking, MasterHunter takes that LeonTokens from stakeholders.

   Using the emergencyWithdraw function stakeholders could get their LP tokens back while LeonTokens will stay on their wallets. Going through this process more and more times, abusers could get minted an unlimited amount of LeonTokens.

   Recommendation: Please do either checking for the LeonTokens amount at the moment of staking or do the LeonTokens safeLeonTransfer enclosed in the try..catch block inside the emergencyWithdraw function.

   Fixed before the second review

3. Flash loan capability exists

   In the provided comments from the customer, it says: "Removed flash loan capability", but checking the LeonicornPairV2's code we've found that the flash loan capability still exists.

   Recommendation: Please either remove the flash loan capability or change the wording that states: "Removed flash loan capability".

   Fixed before the second review

🟩 🟩 Medium

1. Fees setting inconsistency

   Having **txFee** separately specified in each LeonicornPairV2 and also separately in the LeonicornRouterV2 may lead to inconsistency.

   Let's see an example: your DEX have 500 pairs created and you decided to change the **txFee** to 0.15%. What you'll need to do:

1. Get all pairs from the LeonicornFactoryV2
2. For each pair call the LeonicornFactoryV2.setTxFee(pair, 15)
3. Call the LeonicornRouterV2.setTxFee(15)

Definitely, those calls could be programmed and you'll no need to worry about the all contracts were updated. But it's also definitely would cost some gas fees to make it all work, especially if your pairs count grows.

Recommendations: In our vision, it is better to keep the txFee only in one place. Which is this place, relates on how do you see **txFee** could be changed per pair or system-wide:

- For the system-wide **txFee** we'd suggest to put it into the LeonicornFactoryV2's public accessible state variable and access it from both LeonicornRouterV2 and LeonicornPairV2 while doing fee calculations
- For the per-pair **txFee** we'd suggest to store it in the LeonicornPairV2 as it is now, but for the LeonicornRouterV2 calculations call to the pair for asking its **txFee** amount

Customer's comment: Access to txFee is required in LeonicornPairV2 and LeonicornRouterV2 contracts. This operation is done by Owner only, we will use script to update the txFee, any extra gas fee will be on us. The solutions that you suggest will require a contract call for each 'swap' to get the txFee, which attracts extra gas. As 'swap' will be invoked by users, this extra gas burden will be on users, which we don't want.

2. Function add() don't check lpToken

MasterHunter contract has a function add() which has the following statement in the comments:

```
// XXX DO NOT add the same LP token more than once. Rewards will be
messed up if you do.
```

There is definitely an issue with rewards calculations if the same lpToken would be provided more than once.

Recommendation: Please consider using mapping(address => boolean) to check if the provided address was already added previously.

Fixed before the second review

3. Centralization issue

LeonToken has an unlimited minting ability given to the contract creator which allows the contract owner to mint any amount of tokens to any address at any time.

Recommendation: If you want to allow only the MasterHunter to mint tokens please consider either move ownership to the MasterHunter contract implicitly in the constructor or create a token from the MasterHunter, which will automatically set MasterHunter as a contract owner.

Customer's comment: We want to mint around 350M LeonTokens for public sale before we launch DEX. After minting 350M tokens, we will deploy farming/pool contracts MasterHunter, LeonVault and others. Along with MasterHunter leonOwership will be changed to 'MasterHunter' manually. To make the farming/pools work we must change the ownership, ownership change is a mandatory operation from us. Once ownership changed we don't have an option to change it again, so it is a partial Centralization issue, which our users are aware of.

4. Vote power logical issue

   The voting power of delegation is not moved from token sender to token recipient along with the transfer. Current transfer doesn't invoke _moveDelegates when transferring tokens.

   Recommendation: We'd recommend to override the "_transfer" function to also transfer delegates on each invocation from the sender of the funds to the recipient.

   Fixed before the second review by removing governance code.

Low

1. Code repetition

   In the LeonicornRouterV2 there are multiple places where the fees calculations occur. It is much clearer to use the LeonicornLibrary contract which already has functions to do such calculations.

   Recommendation: Please do fees calculations in one place to avoid mistakes.

   Fixed before the second review

2. Unnecessary operations

   When allocPoint is not changed for the pool, there still sub and add functions called on the totalAllocPoint, which just consumes gas doing nothing.

   Recommendation: Please move totalAllocPoint calculation to the *if (poolInfo[_pid].allocPoint != _allocPoint)* block.

   Fixed before the second review

3. No event on critical values change

   There are places in the code where critical values are changed (like txFee, devaddr, minPriceTicketInLeon, maxPriceTicketInLeon, etc.) but no event is emitted. Emitting events on changes is crucial for tracking the contract changes off-chain and providing overall contract visibility to the community.

   Recommendation: Please make sure your contracts are emitting events on changing critical values.

   Fixed before the second review

4. Missing address zero-validation

   The assigned value to "devAddr", "txFeeSetter", etc should be verified as non zero value to prevent being mistakenly assigned as zero-address. Violation of this may cause losing ownership on the parts of contracts.

   Recommendation: Please make sure your contracts are cheking provided addresses to be the correct ones.

   Fixed before the second review

5. Boolean equality

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

   Recommendation: Remove the equality to the boolean constant.

   Fixed before the second review

6. A public function that could be declared external

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

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

   Fixed before the second review

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 **3** high, **4** medium and **6** low severity issues.

After the second review security engineers found that **all** issues were addressed or commented on by the customer.

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.