



CFG NINJA AUDITS

Security Assessment

League Of Hamsters

Token

February 10, 2023

Audit Status



Table of Contents

1 Assessment Summary

2 Technical Findings Summary

3 Project Overview

3.1 Token Summary

3.2 Risk Analysis Summary

3.3 Main Contract Assessed

4 Smart Contract Risk Checks

4.1 Mint Check

4.2 Fees Check

4.3 Blacklist Check

4.4 MaxTx Check

4.5 Pause Trade Check

5 Contract Ownership

6 Liquidity Ownership

7 KYC Check

8 Smart Contract Vulnerability Checks

8.1 Smart Contract Vulnerability Details

8.2 Smart Contract Inheritance Details

8.3 Smart Contract Privileged Functions

9 Assessment Results and Notes(Important)

10 Social Media Check(Informational)

11 Technical Findings Details

12 Disclaimer



Assessment Summary

This report has been prepared for League Of Hamsters Token on the Binance Smart Chain network. CFGNINJA provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.



Project Overview

Token Summary

Parameter	Result
Address	0x7E1218E42E91A78D7A3d817C17c0a036eD54Db43
Name	League Of Hamsters
Token Tracker	League Of Hamsters (LHAR)
Decimals	18
Supply	2,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.7+commit.e28d00a7
Contract Name	LeagueOfHamsters
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x7E1218E42E91A78D7A3d817C17c0a036eD54Db43#code
Payment Tx	Corporate



Project Overview

Risk Analysis Summary

Parameter	Result
Buy Tax	0%
Sale Tax	0%
Is honeypot?	Clean
Can edit tax?	No
Is anti whale?	No
Is blacklisted?	No
Is whitelisted?	No
Holders	0
Confidence Level	High Risk

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.



Project Overview

Simulation Summary

Parameter	Result
Transfer From Owner	Pass
Transfer From Holder	Pass
Add Liquidity	Pass
Buy from Owner	Pass
Buy from Holder	Pass
Remove Liquidity	Pass
SwapAndLiquify	Pass
RemoveLiquidity	Pass
LaunchPad	PinkSale

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.



Main Contract Assessed Contract Name

Name	Contract	Live
League Of Hamsters	0x7E1218E42E91A78D7A3d817C17c0a036eD54Db43	Yes

TestNet Contract Assessed Contract Name

Name	Contract	Live
League Of Hamsters	0xAb1d8AD5BFC00466DC44DD0B33E61B252a5BB977	Yes

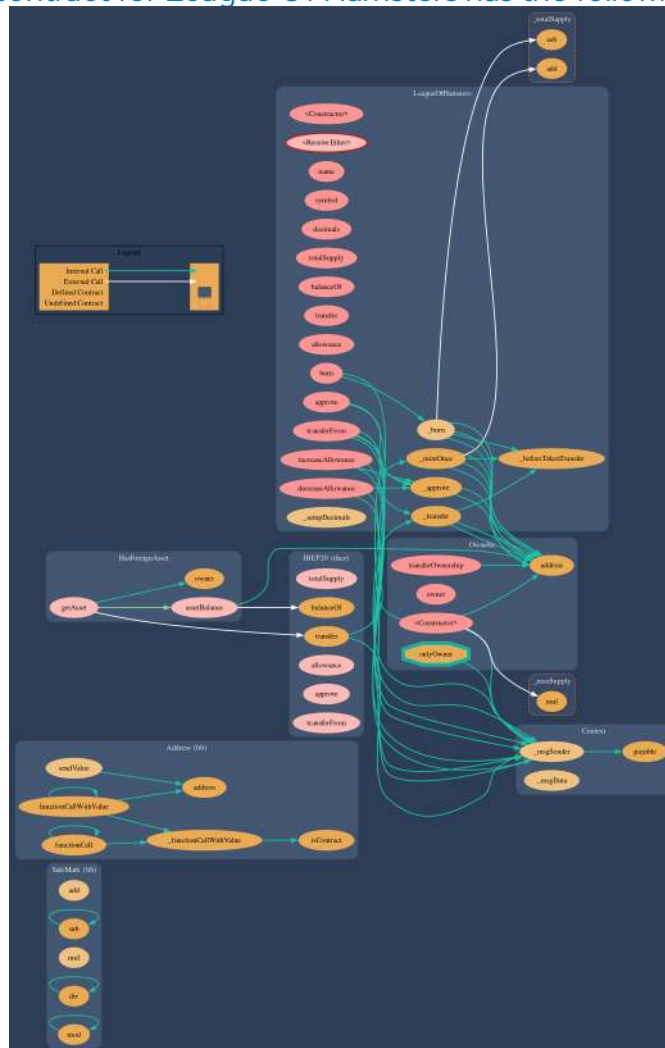
Solidity Code Provided

SolID	File Sha-1	FileName
LeagueOfHamsters	da39a3ee5e6b4b0d3255bfef95601890afd80709	hamster.sol



Call Graph

The contract for League Of Hamsters has the following call graph structure.



KYC Information

The Project Owners of League Of Hamsters have provided KYC Documentation.

KYC Certificated can be found on the Following:
KYC Data

KYC Information Notes:

Auditor Notes:

Project Owner Notes:



Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	hamster.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	hamster.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	hamster.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	hamster.sol	L: 16 C: 0
SWC-104	Pass	Unchecked Call Return Value.	hamster.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	hamster.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	hamster.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	hamster.sol	L: 0 C: 0
SWC-108	Pass	State variable visibility is not set..	hamster.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	hamster.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	hamster.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-111	Pass	Use of Deprecated Solidity Functions.	hamster.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	hamster.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	hamster.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	hamster.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	hamster.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	hamster.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	hamster.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	hamster.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	hamster.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randommness.	hamster.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	hamster.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	hamster.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	hamster.sol	L: 467 C: 23
SWC-124	Pass	Write to Arbitrary Storage Location.	hamster.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-125	Pass	Incorrect Inheritance Order.	hamster.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	hamster.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	hamster.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	hamster.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	hamster.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	hamster.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	hamster.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	hamster.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	hamster.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	hamster.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	hamster.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	hamster.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.



Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to 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.

Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.



Smart Contract Vulnerability Details

SWC-123 - Requirement Violation

CWE-573: Improper Following of Specification by Caller

Description:

The Solidity `require()` construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations. Violations of a requirement can indicate one of two possible issues:

- A bug exists in the contract that provided the external input.
- The condition used to express the requirement is too strong.

Remediation:

If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

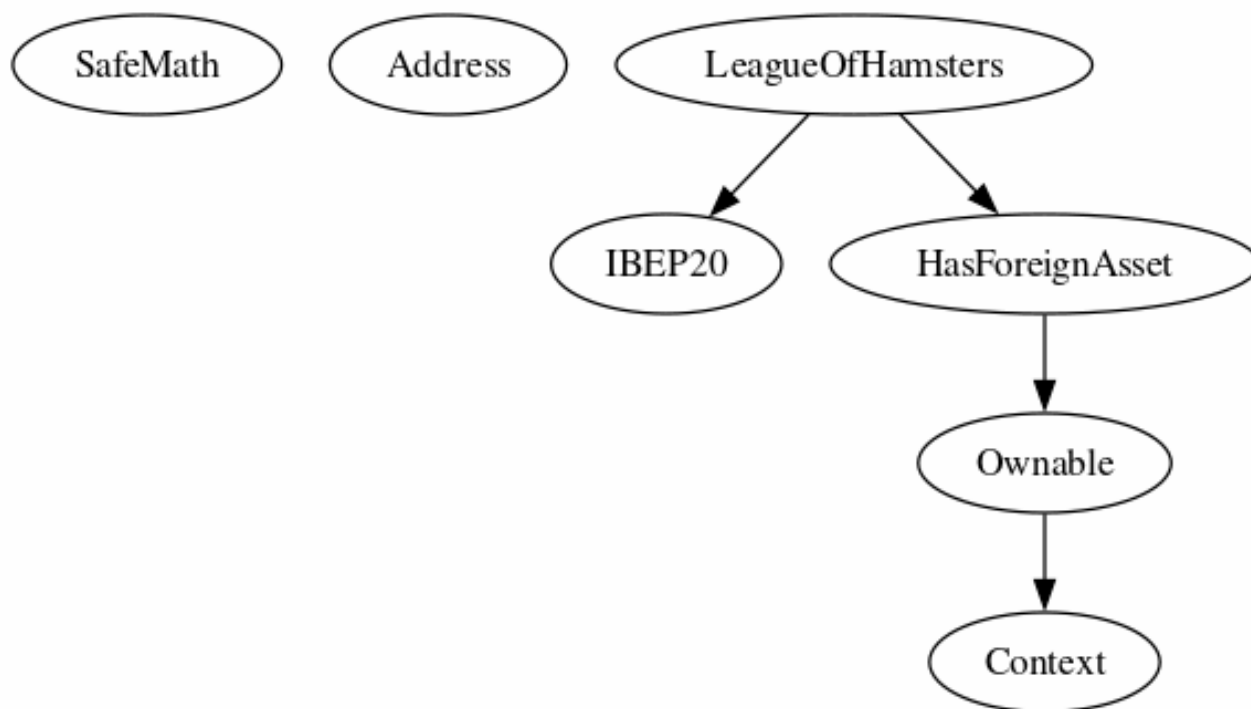
References:

The use of `revert()`, `assert()`, and `require()` in Solidity, and the new REVERT opcode in the EVM








Inheritance

The contract for League Of Hamsters has the following inheritance structure.








Technical Findings Summary

Classification of Risk

Severity	Description
 Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
 Major	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
 Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
 Minor	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
 Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
 Critical	0	0	0
 Major	0	0	0
 Medium	0	0	0
 Minor	1	1	0
 Informational	0	0	0
Total	1	1	0



Smart Contract Advance Checks



ID	Severity	Name	Result	Status
LHAR-01	Minor	Potential Sandwich Attacks.	Fail	Found
LHAR-02	Minor	Function Visibility Optimization	Pass	Not-Found
LHAR-03	Minor	Lack of Input Validation.	Pass	Pending
LHAR-04	Major	Centralized Risk In addLiquidity.	Pass	Not-Found
LHAR-05	Major	Missing Event Emission.	Pass	Not-Found
LHAR-06	Minor	Conformance with Solidity Naming Conventions.	Pass	Not-Found
LHAR-07	Minor	State Variables could be Declared Constant.	Pass	Not-Found
LHAR-08	Major	Dead Code Elimination.	Pass	Not-Found
LHAR-09	Major	Third Party Dependencies.	Pass	Not Found
LHAR-10	Major	Initial Token Distribution.	Pass	Not-Found
LHAR-11	Critical	The use of setHoldTime can lead to a pause trade or honeyPot State	Pass	Not-Found
LHAR-12	Major	Centralization Risks In The X Role	Pass	Not Found
LHAR-13	Informational	Extra Gas Cost For User..	Pass	Not-Found
LHAR-14	Medium	Unnecessary Use Of SafeMath	Pass	Not-Found



ID	Severity	Name	Result	Status
LHAR-15	Medium	Symbol Length Limitation due to Solidity Naming Standards.	Pass	Not-Found
LHAR-16	Medium	Invalid collection of Taxes during Transfer.	Pass	Not-Found



LHAR-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	 Minor	hamster.sol: 0,0	 Found

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()
- addLiquidityETH()

Remediation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

References:

What Are Sandwich Attacks in DeFi – and How Can You Avoid Them?.



Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/leagueofhamster	Pass
Other	https://discord.gg/Uq3exZhw	Pass
Website	https://leagueofhamsters.games/	Pass
Telegram	https://t.me/leagueofhamsters	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:



Assessment Results

Score Results

Review	Score
Overall Score	96/100
Auditor Score	90/100
Review by Section	Score
Manual Scan Score	47/50
SWC Scan Score	35 /37
Advance Check Score	14 /13

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 80 Points, if a project does not attain 80% is an automatic failure. Read our notes and final assessment below.

Audit Passed



Assessment Results

Important Notes:

- No Issues or vulnerabilities were found in the contract.
- the contract is developed by Roman.
- the customer has KYC and Doxxed.
- <https://leagueofhamsters.games/#team>

Auditor Score =90

Audit Passed



Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria and is readable by all the developers.



Disclaimer

CFGNINJA has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocacy for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and CFGNINJA is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will CFGNINJA or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

