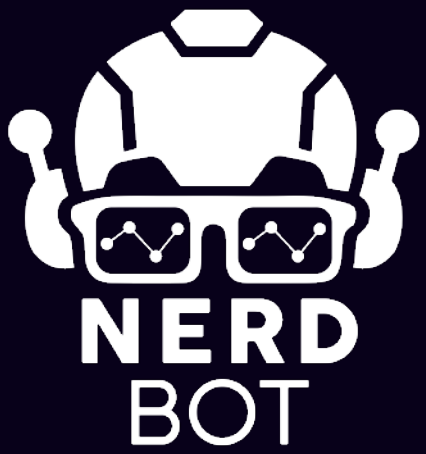


SMART CONTRACT REVIEW

zokyo

October 17th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING NERD LABS

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- Medium issues: 1 issue (acknowledged) = 0 points deducted (since it's not impacting business logic).
- Low issues: 4 issues (all resolved/fixed) = -2 points each (considering the importance of fixing) = -8 points total.
- Informational issues: 2 issues (all resolved) = -0.5 point each = -1 point total.
Thus, 100 - 8 - 1 = 91

However, with 99,36% test coverage (above the industry standard of 95%) we'll add +4 points (this is arbitrary and based on the importance placed on good test coverage).

So, 91 + 5 = 96.

TECHNICAL SUMMARY

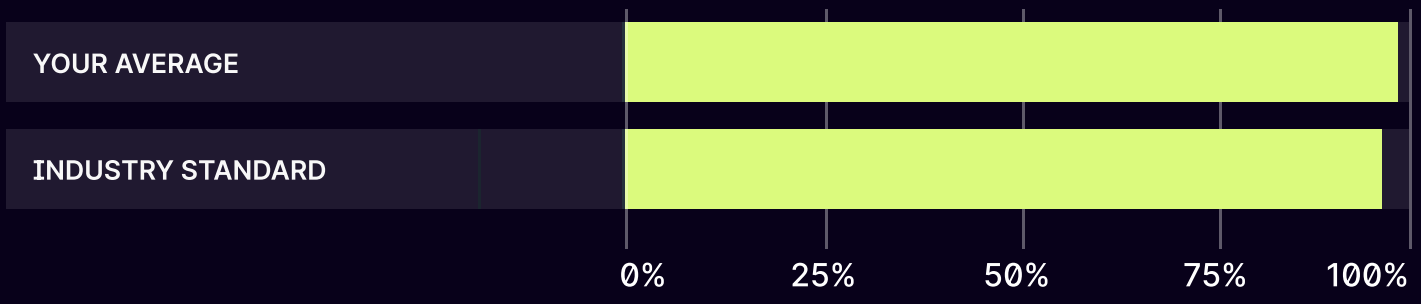
This document outlines the overall security of the Nerd Labs smart contract evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Nerd Labs smart contract codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



99,36% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Nerd Labs team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Nerd Labs repository:
Repo: https://github.com/thedatanerd100/nerd-token

Last commit: ab324b953b339fddfc9e32b040314e053a3d1ab3

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- NerdToken.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Nerd Labs smart contract. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Foundry testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

The contract exhibits excellent writing and structure. Our audit revealed one issue of medium severity, four of low severity, and two informational issues, with no critical or high-severity concerns identified. These findings are elaborated upon in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Nerd Labs team and the Nerd Labs team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Possibility of Slippage attacks	Medium	Acknowledged
2	Missing max limit for fees	Low	Resolved
3	Missing zero address check	Low	Resolved
4	No Validation on the `feeWallet` Address	Low	Resolved
5	Ownership-Based Function Controls	Low	Acknowledged
6	Lack of Events for Critical Changes	Informational	Resolved
7	Redundant Check for Zero Address in `_transfer` Function	Informational	Resolved

Possibility of Slippage Attacks

No slippage has been set during swapping of tokens in the function **swapTokensForEth()**. This could lead to the user being vulnerable to slippage attacks in which an attacker can manipulate the prices and frontrun and backrun the user such that the user gets much lesser amount of tokens than expected. See **line: 823** for the same.

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
```

This happens due to the fact that `0` value has been passed for the amountOutMin parameter of the **swapExactTokensForETHSupportingFeeOnTransferTokens() function.**

Recommendation:

In order to fix this issue, it is advised to set proper slippage value. This would mean dynamically calculating the **amountOutMin** parameter by letting the user specify the slippage value/percentage that they want to use. Note that setting a hardcoded value here for slippage could result in freezing of users funds in times of high volatility.
Refer- https://dacian.me/defi-slippage-attacks

LOW-1 | RESOLVED

Missing Max Limit for Fees

There is missing sanity checks for function parameter **_buyFee** and **_sellFee** of the **updateFees()** function. The **_buyFee** and **_sellFee** can be set arbitrarily high, which can result in almost all of the amount(on which the fees is levied) being paid in fees.

Recommendation:
It is advised to add a max limit for buyFee and sellFee.

LOW-2 | RESOLVED

Missing Zero Address Check

Missing zero address checks for parameter pair of the function **setAutomatedMarketMakerPair()**. This could lead to zero address being accidentally set to true for **automatedMarketMakerPairs[]** mapping and can lead to unintended issues.

Recommendation:
It is advised to add a zero address require check for the same.

LOW-3 | RESOLVED

No Validation on the `feeWallet` Address

Description: The `_updateFeeWallet` function allows setting the `feeWallet` without any validation if it's a valid address or not. It's possible to set it to the zero address or another potentially harmful address.
An incorrect fee wallet address, like the zero address, could cause funds to become irretrievable.

Recommendation:
Implement checks to validate the provided `feeWallet` address before setting it.

LOW-4 | ACKNOWLEDGED

Ownership-Based Function Controls

Description: Functions like `enableTrading`, `removeLimits`, and `updateSwapEnabled` are controlled solely based on contract ownership. If the owner's address is compromised, it could lead to catastrophic consequences.

If the owner's address is compromised, an attacker could halt trading, change critical contract parameters, or even drain funds.

Recommendation:
Implement a multi-signature or DAO-based control for critical functions rather than relying solely on the `onlyOwner` modifier.

INFORMATIONAL-1 | RESOLVED

Lack of Events for Critical Changes

Description: Some critical state changes, like `updateSwapEnabled` or `enableTrading`, don't emit events. This makes it harder for external systems to track the contract's actions and state changes.

Recommendation:
Emit descriptive events for all state-changing functions in the contract.

Redundant Check for Zero Address in `_transfer` Function

Description:
The `_transfer` function in the provided in nerd token contains a redundant check for the zero address. Initially, the function checks if the to address is the zero address using a require statement. However, further into the function, there's another check for the zero address within a compound `if` condition.

Recommendation:
Remove the redundant check for the zero address in the compound `if` statement. The initial `require` statement is sufficient to prevent transfers to the zero address.

	NerdToken.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Nerd Labs in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Foundry testing framework.

The tests were based on the functionality of the code, as well as a review of the Nerd Labscontract requirements for details about issuance amounts and how the system handles these.

Running 15 tests for test/NerdToken.t.sol:ERC20Test
[PASS] testApprove() (gas: 40284)
[PASS] testBurn() (gas: 21893)
[PASS] testDecimals() (gas: 5593)
[PASS] testDecreaseAllowance() (gas: 44035)
[PASS] testERC20Transfer() (gas: 40436)
[PASS] testERC20TransferBranchA() (gas: 13186)
[PASS] testERC20TransferBranchB() (gas: 9247)
[PASS] testERC20TransferBranchC() (gas: 11447)
[PASS] testIncreaseAllowance() (gas: 35812)
[PASS] testMint() (gas: 40089)
[PASS] testMsgData() (gas: 6231)
[PASS] testName() (gas: 9903)
[PASS] testRevertBurnDuetoAddressZero() (gas: 8881)
[PASS] testRevertBurnDuetoLessBalance() (gas: 14352)
[PASS] testSymbol() (gas: 9968)
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 7.38s

Running 3 tests for test/NerdToken.t.sol:OwnableTest
[PASS] testOwner() (gas: 7744)
[PASS] testRenounceOwnership() (gas: 16173)
[PASS] testTransferOwnership() (gas: 22558)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 7.73s

Running 29 tests for test/NerdToken.t.sol:NerdTokenTest
[PASS] testEnableTrading() (gas: 18735)
[PASS] testExcludeFromFees() (gas: 30118)
[PASS] testFeelessTransfer() (gas: 54865)
[PASS] testRemoveLimits() (gas: 12760)
[PASS] testRevertBadDeployment() (gas: 2850530)

```
[PASS] testRevertDuetoLessAllowanceOnTransfer() (gas: 97124)
[PASS] testRevertFromZeroAddressTransfer() (gas: 9362)
[PASS] testRevertToZeroAddressTransfer() (gas: 9397)
[PASS] testRevertWithdrawStuckEth() (gas: 63314)
[PASS] testRevertZeroTransfer() (gas: 21072)
[PASS] testSetAutomatedMarketMakerPair() (gas: 41448)
[PASS] testSetPreMigrationTransferable() (gas: 59164)
[PASS] testSetUp() (gas: 7703)
[PASS] testSuccessfulDeployment() (gas: 4531048)
[PASS] testTransferRevertDuetoPremigration() (gas: 89555)
[PASS] testTransferWithFeeA() (gas: 94253)
[PASS] testTransferWithFeeB() (gas: 152808)
[PASS] testTransferWithFeeC() (gas: 174379)
[PASS] testTransferWithRemovedLimits() (gas: 53837)
[PASS] testTransferWithSwapA() (gas: 235479)
[PASS] testTransferWithSwapB() (gas: 250847)
[PASS] testTransferWithTradingNotActive() (gas: 136168)
[PASS] testUpdateFeeWallet() (gas: 16756)
[PASS] testUpdateFees() (gas: 19678)
[PASS] testUpdateSwapEnabled() (gas: 16030)
[PASS] testUpdateSwapTokensAtAmount() (gas: 27323)
[PASS] testWithdrawStuckEth() (gas: 43772)
[PASS] testWithdrawStuckNerd() (gas: 37110)
[PASS] testWithdrawStuckToken() (gas: 104468)
Test result: ok. 29 passed; 0 failed; 0 skipped; finished in 10.84s

Ran 3 test suites: 47 tests passed, 0 failed, 0 skipped (47 total tests)
```

We are grateful for the opportunity to work with the Nerd Labs team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Nerd Labs team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo