Mantisec Labs

Smart Contract Audit

`KeywordStaking.sol`
Presearch

July 2025

Mantisec Labs

Contents

Disclaimer

This disclaimer is to inform you that the report you are reading has been prepared by Mantisec Labs for informational purposes only and should not be considered as investment advice. It is important to conduct your own independent investigation before making any decisions based on the information contained in the report. The report is provided "as is" without any warranties or conditions of any kind and Mantisec Labs excludes all representations, warranties, conditions, and other terms. Additionally, Mantisec Labs assumes no liability or responsibility for any kind of loss or damage that may result from the use of this report.

It is important to note that the analysis in the report is limited to the security of the smart contracts only and no applications or operations were reviewed. The report contains proprietary information, and Mantisec Labs holds the copyright to the text, images, photographs, and other content. If you choose to share or use any part of the report, you must provide a direct link to the original document and credit Mantisec Labs as the author.

By reading this report, you are accepting the terms of this disclaimer. If you do not agree with these terms, it is advisable to discontinue reading the report and delete any copies in your possession.

Audit Process & Methodology

The Mantisec Labs team carried out a thorough audit for the project, starting with an in-depth analysis of code design patterns. This initial step ensured the smart contract's architecture was well-structured and securely integrated with third-party smart contracts and libraries.Also, our team conducted a thorough line-by-line inspection of the smart contract,seeking out potential issues such as Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, among others.

During the Unit testing phase, we assessed the functions authored by the developer to ascertain their precise functionality. Our Automated Testing procedures leveraged proprietary tools designed in-house to spot vulnerabilities and security flaws within the Smart Contract. The code was subjected to an in-depth audit administered by an independent team of auditors, encompassing the following critical aspects:

- Scrutiny of the smart contract's structural analysis to verify its integrity.
- Extensive automated testing of the contract
- A manual line-by-line Code review, undertaken with the aim of evaluating, analyzing, and identifying potential security risks.
- An evaluation of the contract's intended behavior, encompassing a review of provided documentation to ensure the contract conformed to expectations.
- Rigorous verification of storage layout in upgradeable contracts.
- An integral component of the audit procedure involved the identification and recommendation of enhanced gas optimization techniques for the contract

Audit Purpose

Mantisec Labs was hired by the Presearch team to review their smart contract.
This audit was conducted in July 2025.

The main reasons for this review were:

- To find any possible security issues in the smart contract.
- To carefully check the logic behind the given smart contract.

This report provides valuable information for assessing the level of risk associated with this smart contract and offers suggestions on how to improve its security by addressing any identified issues.

Contract Details

Project Name	Presearch
Contract links	`KeywordStaking.sol`
Language	Solidity
Type	ERC20

Security Level Reference

Each problem identified in this report has been categorized into one of the following severity levels:

- **Critical**: Vulnerabilities that present an immediate and serious threat to system or data integrity, demanding urgent action.
- **High**: Significant risks that have the potential to cause major security breaches or loss of functionality.
- **Medium**: Issues that moderately affect system performance or security and require timely resolution.
- **Low**: Low-risk concerns primarily related to optimization and code quality, with minimal direct impact on system security.
- **Informational**: Observations or recommendations that do not pose any direct risk but provide insights for potential improvements or best practices.

Severity	Score
Critical	4-5
High	3-4
Medium	2-3
Low	1-2
Informational	0-1

Findings Overview

Contract Names:

- `KeywordStaking.sol`

Critical	High	Medium	Low	Informational
0	0	4	2	0

Issue	Severity	Fix Date
M01- Missing Emergency Pause Mechanism	**Medium**(2)	10-07-2025
M02- Signature Hash Not Fully Typed	**Medium**(2)	10-07-2025
M03- No Domain Separation in Signatures	**Medium**(2)	10-07-2025
M04- Expensive Loops in unstake() for Large User Sets	**Medium**(2)	21-07-2025
L01- Unbounded batchId String Length	**Low**(1)	10-07-2025
L02- Unsafe Token Transfers Without SafeERC20	**Low**(1)	21-07-2025

Findings Details

`KeywordStaking.sol`

M01- Missing Emergency Pause Mechanism

Severity: Medium
Impact: Operational Risk, Emergency Response Deficiency

Status: Patched

Issue:

The KeywordStaking contract lacks a pause() mechanism to halt critical functionality such as staking, unstaking, and batch migrations in case of emergencies, exploits, or misconfigurations. This limits the admin's ability to respond to unforeseen issues and increases risk to user funds.

Affected Functions: stake(...), unstake(...), migrateBatchKeywordStakes(...)

Recommendation:

Integrate the Pausable contract from OpenZeppelin and apply the whenNotPaused modifier to key state-changing functions. Additionally, expose pause() and unpause() functions restricted to the contract admin.

M02- Signature Hash Not Fully Typed (abi.encodePacked Used)

Severity: Medium
Impact: Potential signature collision or malleability in rare edge cases

Status: Patched

Issue:

The contract uses abi.encodePacked when generating the message hash for signatures in _verifyStakeSignature and _verifyBatchSignature. This can lead to ambiguous encoding for dynamic types (like string) and may create collisions if not used carefully.

```
bytes32 messageHash = keccak256(
    abi.encodePacked(msg.sender, amount, nodeId, batchId, deadline, nonce)
);
```

If two different sets of parameters result in the same packed byte sequence (especially due to dynamic types like string), an attacker might forge a valid signature for unintended data.

Recommendation:

Use abi.encode instead of abi.encodePacked for unambiguous encoding.

M03- No Domain Separation in Signatures

Severity: Medium
Impact: Cross-contract or cross-chain replay attacks

Status: Patched

Issue:

The contract constructs ECDSA message hashes without including a domain separator (e.g., chainId, contract address). As a result, a valid signature could potentially be replayed:

- on a different contract with the same message format
- on another chain with the same msg.sender

Impact Scenario:

If two chains or contracts share the same signer logic, an off-chain signature intended for one can be reused on the other, unless additional checks are in place.

Recommendation:

Add domain separation by including address(this) and block.chainid in the signed message.

M04- Expensive Loops in unstake() for Large User Sets

Severity: Medium
Location: unstake()

Status: Patched

Description:

The unstake() function contains two unbounded for loops:

- One loop removes the user from keywordInfo.stakers
- Another removes the keyword from _userKeywords[msg.sender]

These loops scale linearly with the number of stakers or keywords, which poses a gas limit risk as the platform grows.

Impact:

- Denial of Service (DoS): Users with many stakes or popular keywords may be unable to unstake due to out-of-gas errors.
- Poor scalability: Limits contract usage as staker count increases.

Recommendation:

Refactor to use index-tracking mappings for O(1) removals and replace for loops with swap-and-pop using stored indexes.

Also:

stakers[] is redundant:

Redundant Because:

- stakes already maps address => Stake.
- Each stake already contains the owner address (which is redundant too — key already gives that).
- You can loop over all stakes *if* you had the keys — which is the purpose of stakers[].

But Keeping stakers[] Has Downsides:

- Adds gas every time a new user stakes.
- Adds gas and complexity when unstaking (loop to remove user).

L01- Unbounded batchId String Length

Severity: Low (Performance-focused)
Impact: Increased Gas Cost, Potential Denial-of-Service (DoS) Vector

Status: Patched

Issue:

The contract uses string for batchId in KeywordMigration and related mappings (_usedBatchIds). This introduces unnecessary gas overhead due to dynamic string handling, especially in mappings and comparison operations (_usedBatchIds[batchId]).

Risk:

String comparisons are more expensive than fixed-size types like bytes32, and if exploited (e.g., by passing large or varied strings), could lead to gas exhaustion and DoS attacks on the migration process.

Recommendation:

Replace string with bytes32 for batchId to reduce gas costs and simplify validation.

Additional Benefit:

Using bytes32 enables deterministic formatting (e.g., keccak256 hashes) and can improve indexing if events are emitted with batch IDs.

Mantisec Labs

L02- Unsafe Token Transfers Without SafeERC20

Severity: Low
Location: stake(), unstake(), migrateBatchKeywordStakes(), recoverTokens()

Status: Patched

Description:

Although try/catch blocks are used to handle ERC-20 transfer reverts, they do not protect against tokens that fail silently by returning false—a behavior observed in tokens like USDT and BNB. As a result, the contract may assume a successful transfer when it actually failed.

Recommendation:

Adopt OpenZeppelin's latest SafeERC20 library for all token operations to enforce proper success checks. Optionally, try/catch can still be used around SafeERC20 calls to guard against unexpected reverts (e.g., from underlying system calls), but SafeERC20 must be the primary enforcement mechanism.

Mantisec Labs

Concluding Remarks

To wrap it up, this audit has given us a good look at the contract's security and functionality.
Our auditors confirmed that all the issues are now resolved by the developers.