SECUREWEI

SECURITY AUDIT REPORT

Prepared by
SECUREWEI

Prepared for
OMNI_LAUGH

Don't trust, verify.

Table of Contents

Summary

About Omni_Laugh

Omni_Laugh is a next-generation meme token engineered for real-world utility, cross-chain interoperability and controlled deflation. Leveraging ZetaChain's UniversalTokenCore, Omni_Laugh exists natively on EVM chains and Zeta's Universal environment, enabling seamless token transfers between networks while applying a modest transaction tax that fuels long-term sustainability.

Security Assessment

The security audit of the Omni_Laugh smart contract was conducted over a four-day period, concluding on July 8, 2025. During the assessment, our team employed comprehensive manual review to rigorously examining both Universal.sol and Ethereum.sol implementations. The audit focused on key security areas, including access control mechanisms, token issuance and deflationary tax logic, potential reentrancy risks, cross-chain transfer safeguards, and compliance with relevant token standards. All critical components and interactions within the Omni_Laugh Token contracts were given special attention to ensure robust security and adherence to best practices.

Over the course of the audit, we identified a total of five security issues:

- 1 **High** severity vulnerability.
- 4 **Info** severity vulnerabilities.

Scope

The audit focused exclusively on the Omni_Laugh smart contract code located at https://github.com/laugh-meme/Omni_Laugh_Universal (commit 00e4abb1a6c21bc92b4c4dd65d7aadcaf0e2abe6). Only the on-chain contract code was assessed; no frontend, server infrastructure, or off-chain components were in scope.
The following files were in scope:

- Universal.sol
- Ethereum.sol

Type: Solidity

Platform: ZetaChain & Ethereum Network

Disclaimer

The findings, observations, and conclusions detailed in this audit report are the result of a comprehensive and diligent assessment performed by experienced security professionals. This assessment was conducted in accordance with recognized industry standards and best practices, and reflects the professional judgment of the audit team based on the scope, information, systems, and documentation made available at the time of the engagement.

While every effort has been made to identify and communicate relevant vulnerabilities, limitations inherent in any security review must be acknowledged. No security assessment, regardless of its depth or methodology, can guarantee the complete absence of vulnerabilities or provide an absolute assurance of security. The dynamic nature of software development and the evolving threat landscape mean that new risks may emerge and system conditions may change after the completion of our work.

This report should therefore be regarded as a significant contribution to a broader, ongoing security posture. It is intended to inform risk-aware decision making and support responsible security practices, but it is not a substitute for comprehensive internal controls, secure development practices, or continued vigilance.

Severity Criteria

Severity	Description
High	Critical vulnerabilities where core functionalities of the protocol or funds are at significant risk.
Major	Vulnerabilities that may affect certain financial aspects or cause specific functionalities to malfunction.
Low	Issues without direct financial impact but could affect user experience or minor protocol operations.
Info	Suggestions for improvements such as gas efficiencies, adherence to best practices, or optimizations.
Likelihood	**Description**
High	The vulnerability can be exploited with minimal or no specific conditions.
Major	Exploitation requires certain conditions to be met or may not present sufficient incentives to be exploited.
Low	Exploitation demands rare conditions or incurs financial costs for the attacker.
N/A	Not applicable.

Findings Summary

Severity	Likelihood	Title
High	High	Bypass of Tax Logic via *UniversalTokenCore::transferCrossChain*
Info	N/A	Limited Tax Granularity and Zero-Tax on Small Transfers
Info	N/A	Missing Events for Critical Actions
Info	N/A	*withdrawTaxes()* is blocked while the token is paused
Info	N/A	Variable optimisation

Detailed Findings

Bypass of Tax Logic via *UniversalTokenCore::transferCrossChain*

Severity: High
Likelihood: High
Target: Universal.sol
Status: Acknowledged
ID: 1

Description

The *OmniLaughToken* contract intends to collect a tax on cross-chain transfers by implementing its own *transferCrossChain* function. However, the inherited *UniversalTokenCore* also exposes a public *transferCrossChain* method, which was not overridden or restricted. As a result, users can call *UniversalTokenCore::transferCrossChain* directly and perform cross-chain transfers without triggering the tax logic in *OmniLaughToken::transferCrossChain*.

Impact

Tax revenue loss: Users avoid paying the intended tax percentage on cross-chain transfers, reducing the funds available for the contract owner or ecosystem.

Recommendation

Override the *UniversalTokenCore::_transferCrossChain* function instead:

```
function _transferCrossChain(address destination, address receiver,
uint256 amount) internal override {
    //Calculate and collect tax
    ...
    super._transferCrossChain(destination, receiver, netAmount);
}
```

Limited Tax Granularity and Zero-Tax on Small Transfers

Severity: Info
Likelihood: N/A
Target: Universal.sol
Status: Acknowledged
ID: 2

Description

The current implementation uses a simple percentage (0–10) and integer division to calculate tax:

```
uint256 taxAmount = (amount * taxPercentage) / 100;
```

Because the calculation is based on a denominator of 100, you can only represent whole-number percentages (1%, 2%, …), and for small ***amount*** values this computation may round down to zero, resulting in no tax being collected.

Impact

- Inability to levy fractional percentage fees (e.g., 0.5%).
- Very small transfers incur zero tax, enabling users to bypass fees for low-value transactions.

Recommendation

- Requiere a minimum ***amount*** so that ***amount * taxPercentage / 100*** doesn't get rounded down to 0.
- Use basis points (bps) with a denominator of 10,000 instead of 100. For example:

```
uint16 public taxBps;    // e.g. 150 = 1.50%
uint256 taxAmount = (amount * taxBps) / 10_000;
```

Missing Events for Critical Actions

Severity: Info
Likelihood: N/A
Target: Universal.sol, Ethereum.sol
Status: Acknowledged
ID: 3

Description

The contract defines critical state-changing functions (***setTaxPercentage***, ***withdrawTaxes***, ***mint***, ***pause***, ***unpause***) but does not emit any events when they are called. Additionally, the ***taxPercentage*** and ***taxCollected*** variables are declared internal, so external tools and users cannot directly observe their values without custom view calls.

Impact

- On-chain monitoring tools, block explorers, and end users cannot easily detect or react to changes in tax parameters, withdrawals, minting, or pause status.
- Lack of events reduces transparency and makes it harder to audit contract activity or trigger off-chain alerts.

Recommendation

- Declare ***taxPercentage*** and ***taxCollected*** as public.
- Emit dedicated events for each critical action, for example:
  - ***TaxPercentageUpdated(oldPercentage, newPercentage)*** in ***setTaxPercentage***.
  - ***TaxesWithdrawn(recipient, amount)*** in ***withdrawTaxes***.
  - ***TokensMinted(recipient, amount)*** in ***mint***.
  - ***Paused(account)*** and ***Unpaused(account)*** (or use OpenZeppelin's events) in ***pause***/***unpause***.

withdrawTaxes() is blocked while the token is paused

Severity: Info
Likelihood: N/A
Target: Universal.sol
Status: Acknowledged
ID: 4

Description

When the token is paused, all ERC20 transfers, including the owner's tax withdrawals, are disabled. As a result, calling ***withdrawTaxes()*** will revert while the contract is paused.

Impact

Collected taxes cannot be recovered if the contract is paused, potentially locking up fee funds indefinitely until unpaused.

Recommendation

Allow ***withdrawTaxes()*** to bypass the paused state. For example, use a direct transfer that ignores pause or temporarily unpause, withdraw, then re-pause within the same transaction under owner control.

Variable optimisation

Severity: Info
Likelihood: N/A
Target: Universal.sol
Status: Acknowledged
ID: 5

The ***taxPercentage*** variable is declared as a ***uint256*** but is constrained to values between 0 and 10. Change the type from ***uint256*** to ***uint8***.

Methodology

The audit was conducted using a comprehensive and multi-layered approach to identify security vulnerabilities, logical flaws, and deviations from best practices within the smart contract codebase. The following methodologies were employed:

Manual Code Review

An extensive manual analysis was performed on the smart contract source code. This involved a line-by-line review to assess logic correctness, access control, input validation, gas optimization, and adherence to industry standards. Manual auditing was also used to identify subtle logic bugs, business logic flaws, and edge cases that automated tools may overlook.

Automated Static Analysis

We employed a suite of state-of-the-art static analysis tools to detect known vulnerability patterns and unsafe programming constructs. These tools scanned the contract code for issues.
Findings from static analysis were carefully reviewed and validated to eliminate false positives and ensure contextual relevance.

Fuzz Testing

Fuzzing techniques were applied to the smart contracts to simulate a wide range of randomised inputs and edge cases. This dynamic analysis aimed to uncover unexpected behaviour, such as assertion failures, state inconsistencies, or crashes that may occur under irregular transaction sequences or malicious inputs.

Threat Modeling and Risk Assessment

We analysed the smart contracts within the broader context of their intended ecosystem and business logic to identify potential threat vectors. Scenarios such as front-running were considered to assess the system's robustness against known attack vectors.

About SecureWei

SecureWei is a blockchain security firm dedicated to safeguarding decentralized systems through rigorous smart contract auditing and protocol assessments. Our mission is to empower developers and projects with the confidence that their code is secure, reliable, and aligned with industry best practices.

At SecureWei, we combine deep domain expertise with cutting-edge analysis techniques to uncover hidden vulnerabilities and design flaws before they can be exploited. Our team brings together professionals with backgrounds in cybersecurity, formal verification, and decentralized application development, enabling us to deliver high-impact, actionable security insights.

Why Choose SecureWei?

- Thoroughness: Every line of code is reviewed by experienced auditors to uncover subtle bugs and complex attack vectors.
- Trust: Our reputation is built on integrity, transparency, and consistently high standards.
- Collaboration: We provide detailed reports and work hand-in-hand with teams to remediate issues effectively.
- Future-Focused: We stay ahead of the curve with evolving blockchain standards and security trends.

Contact

To learn more about our services or to request an audit, feel free to reach out:

- Twitter (X): @securewei
- Telegram: @securewei
- Website: securewei.com

SecureWei is committed to raising the security bar in Web3.