verichains

SECURITY AUDIT OF

FORU AI TOKEN

Public Report

Aug 18, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Aug 18, 2025. We would like to thank the ForU AI for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the ForU AI Token. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified a vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About ForU AI Token

About **ForU AI**: The Future of **AI Agents** for Communities**ForU AI** is revolutionizing the way AI agents serve communities with ForU AI's **Real-World AI Agent Launchpad**, powered by ForU AI's cutting-edge Real-World AI Engine. This engine consists of an AI Network with three key components: User **AI-DIDs**, Community **AI-DIDs**, and **Community AI Agents**.

ForU AI's mission is to bring ideas, fandoms, communities, hobbies, and interests on-chain, enabling users to create intelligent, autonomous AI agents that cater to their needs—while also ensuring their financial interests are integrated.

While AI agent launchpads are not new, ForU AI's innovation lies in the intelligence embedded within ForU AI's **AI-DIDs**, even before the AI agents are deployed. The result? A powerful ecosystem where you can build an army of AI agents **For U**, pushing beyond yForU AI's limits like never before.

More infomations at ForU AI's social media.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the ForU AI Token. It was conducted on commit `9d3ea5450086aea99a14668c45ff99da67b7454f` from git repository: *https://github.com/untukmuai/foruai-token-solidity*

SHA256 Sum	File
`a5b5252b400720f3311480138f36956ec2acede9c15ab85ad66011cb7440154a`	`./ForU.sol`
`3d59dbd01ebce9e5480189565bd5cc4623d31efc26b7217106a96ff898e2d221`	`./UUPSUpgradeableProxy.sol`

Contracts are implemented Sei mainnet at address: `0x04f0542fb2ec871A6e28156039d71fD9a9e85a92`. The details of the deployed smart contract are listed in the table below:

PROPERTY	VALUE
Token name	`ForU AI` FORU
Owner	`0x19273af51d1ee8e2c1babc55cc1de975e2945901`

PROPERTY	VALUE
Contract Implementations	ForUV2.sol
Creator	0xD802b8a9C30A67BD6F06A6757C95fEFf2c617D0c
Compiler Version	v0.8.28+commit.7893614a
Contracts address	./UUPSUpgradeableProxy.sol: 0x04f0542fb2ec871A6e28156039d71fD9a9e85a92
	./ForUV2.sol: 0x529EDc4CD7D7FF773837Ce0FF5492d1Fd2d0eafd
Explorer	*https://seitrace.com/address/0x04f0542fb2ec871A6e28156039d71fD9a9e85a92?chain=pacific-1*

Table 1. ForU AI Token's properties

Note: The scope of the audit is limited to the source code files provided. All contracts in the scope are **upgradeable** contracts, the contract owner can change the contract logic at any time in the future.

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit

- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.4. Disclaimer

ForU AI acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. ForU AI understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, ForU AI agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the ForU AI will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the ForU AI, the final report will be considered fully accepted by the ForU AI without the signature.

2. AUDIT RESULT

2.1. Overview

The ForU AI Token was written in `Solidity` language, with the required version to be `^0.8.28` for `UUPSUpgradeableProxy.sol` and `^0.8.27` for `ForU.sol`.

The token contract extends `UUPSUpgradeable`, `OwnableUpgradeable`, `ERC20Upgradeable`, and `ERC20PermitUpgradeable` from the `OpenZeppelin` library, while the proxy contract extends `ERC1967Proxy`:

- The token contract is designed to be **upgradeable**, so it doesn't use a constructor for initialization. Instead, it uses `_disableInitializers` to prevent unwanted re-initialization after deployment.
- The `onlyAdmin` modifier restricts access to specific functions, allowing only the admin (configured via the upgradeable proxy system) to execute them. The function uses `ERC1967Utils.getAdmin` to get the admin's address. Only the owner can change the admin through `changeAdmin`. Additionally, the `_authorizeUpgrade` function of `UUPSUpgradeable` is overridden to ensure only the admin can call it.
- The `emergencyWithdraw` function lets the admin withdraw both ETH and ERC-20 tokens from the contract in emergency situations. This function is admin-restricted to prevent unauthorized withdrawals. This contract now has `receive() external virtual payable {}` to receive ETH.

2.2. Findings

#	Title	Severity	Status
1	`emergencyWithdraw` contains a logical error when withdrawing all ETH	LOW	Fixed

2.2.1. LOW - `emergencyWithdraw` contains a logical error when withdrawing all ETH.

Positions:

- *contract/ForU.sol*#L48

Description:

In the `emergencyWithdraw` function, the admin can withdraw a specified token amount during emergencies. However, when `_token == address(0)`, the function transfers the entire contract balance (`address(this).balance`) instead of the specified amount.

```
function emergencyWithdraw(address _token, address _recipient, uint256 _amount) external
virtual onlyAdmin {
    if (_token == address(0)) {
        require(address(this).balance >= _amount, "Insuf balance");
        payable(_recipient).transfer(address(this).balance);
    } else {
        uint256 contractBalance = uint256(IERC20Metadata(_token).balanceOf(address(this)));
        require(contractBalance >= _amount, "Insuf token balance");
        IERC20Metadata(_token).safeTransfer(_recipient, _amount);
    }

    emit EmergencyWithdraw(_token, _recipient, _amount);
}
```

RECOMMENDATION

The transfer statement should use `amount` instead of `address(this).balance`.

UPDATES

- *14 Feb,2025*, The team has acknowledged the issue and has implemented the recommended changes.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Feb 12, 2025*	Public Report	Verichains Lab
1.1	*Feb 14, 2025*	Public Report	Verichains Lab
1.2	*Aug 18, 2025*	Public Report	Verichains Lab

Table 3. Report versions history