LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For JU SWAP

15 April 2025

lunaray.co

Table of Contents

1. Overview

On April 3, 2025, the Lunaray Security Team's security team received a security audit request for the **JU SWAP projec**t. The team completed the audit on April 15, 2025 for the **JU SWAP contract**. During the audit process, security audit experts from Lunaray's security team maintained communication and ensured information symmetry with the JU SWAP project owner, conducted the security audit with controlled operational risks, and avoided risks to project generation and operations during the testing process.

Through communication and feedback with the JuChain Bridge project, we confirmed that the vulnerabilities and risks found during the audit have been fixed or are within acceptable limits. The results of this audit of JU SWAP smart contract.

contract security audit: **Passed**

Audit Report Hash:
1424B393D6CDE486B610ED33DFDC6AF7529763B8046009BC37BEAB71AFD2A2D2

2. Background

2.1 Project Description

Project name	Ju Swap
Contract type	Decentralized Finance
Code language	Solidity
Public chain	JuChain
Project website	https://testnet.juchain.org/
Introduction	JuChain, a modular blockchain built on OP Stack with Layer 2 consensus, delivers high performance. Its core cross-chain bridge enables seamless multi-chain asset/data transfers through decentralized services, featuring minimal fees and rapid operations. Leveraging modular architecture, the solution offers simplified interfaces for broader accessibility while providing infrastructure for DeFi expansion.
Contract file	02JUFactory.sol 03JURouter02.sol Tudst.sol USDT.sol WJU.sol usdtbsc.sol

2.2 Audit Range

2.2.1 Smart contract file name and corresponding SHA256

Name	SHA256
02JUFactory.sol	1968E677988428461047391E4448CAF5E86D06643540F7AD1B4ED59EDC6933E4
03JURouter02.sol	05C25FF7E6DC204AC46D441A91FC9B5ED1C3F95E8DEF14F0B8866030D735F2F5
Tudst.sol	AABE375F325D3BA6AFE7CF6DDA775C75AD74BDAA838321A62987D1B790BE4F37
USDT.sol	265D8FB1A820B07CE14AFEF55C90F3DE1CF3C127BB482C63E4D8132971A02E39
WJU.sol	2F177A5F3081BE7EAAD1C7701B89A5AB78D4D732254D36991E8D084F1CEADF99
usdtbsc.sol	37AA1CFC8228B55FC7E192E3FD4B14BC78C047D7D3063F2A8A254334F54B583B

3. Project contract details

3.1 Contract Overview

WJU Contract

This contract is a wrapper protocol that enables bidirectional conversion between native tokens (such as Ethereum's ETH) and ERC20-standard tokens. Users can deposit native tokens into the contract and exchange them 1:1 for an ERC20 token called "Wrapped JU" (WJU), while also supporting the burning of WJU to reclaim the corresponding native assets. The contract automatically mints an equivalent amount of WJP to the user's address through the deposit function when receiving native tokens sent by users, and provides a withdraw function allowing users to burn WJP and redeem native tokens. As an ERC20-compliant token, the contract implements transfer (transfer), approval (approve), and delegated transfer (transferFrom) functionalities, enabling users to freely transfer WJP among themselves or authorize other applications to manage their tokens.

02JUFactory Contract

This contract group implements a complete decentralized trading protocol, with core functionality centered around an Automated Market Maker (AMM) mechanism. It allows users to permissionlessly create trading pairs for any two ERC20 tokens and provide liquidity, while also enabling token swaps through liquidity pools.

JUV2Factory, as the factory contract, is responsible for creating and managing all trading pair contracts. By invoking createPair, it generates new JUV2Pair instances, ensuring each token pair corresponds to a unique liquidity pool. It also allows protocol administrators to set a fee recipient address for revenue distribution.

JUV2Pair, as the specific trading pair contract, manages the liquidity pool for two tokens. Users can deposit equivalent assets via mint to receive ERC20-standard liquidity provider (LP) tokens or redeem assets by burning LP tokens through burn. Its built-in

swap function executes token swaps based on the constant product formula (reserve0 * reserve1 = k) and charges a 0.3% fee per transaction (part of which is converted into protocol revenue via _mintFee, minting LP tokens to a designated address).

JUV2ERC20, as the underlying implementation of liquidity provider tokens, extends standard ERC20 functionality with off-chain signature-based approvals (permit), allowing users to authorize third parties to manage their LP tokens via signatures. It also enhances signature security by adhering to the EIP-712 standard through DOMAIN_SEPARATOR.

The protocol also integrates an oracle feature, recording cumulative prices per block (price0CumulativeLast and price1CumulativeLast) to enable external contracts to calculate Time-Weighted Average Prices (TWAP). Functions like skim and sync ensure reserve balances match actual token balances to handle irregular transfers.

The system employs a lock modifier to prevent reentrancy attacks in critical operations, relies on math libraries (SafeMath, UQ112x112) for precise calculations, and supports flash loans (via the JUV2Call callback for uncollateralized borrowing). Ultimately, it establishes a decentralized, self-governing trading infrastructure with a sustainable fee revenue model.

03JURouter02 Contract

The JUV2Router02 contract, as the core router of a decentralized exchange, primarily manages user liquidity and provides token swap services. By integrating with a factory contract, it automatically creates trading pairs, allowing users to add or remove liquidity for any two tokens (including ETH in its WETH-wrapped form) and generates corresponding liquidity receipt tokens. It supports various swap methods, including exact input/output token exchanges, direct ETH-to-token swaps, and is specifically adapted for tokens with transfer fee mechanisms, ensuring accurate received amounts through real-time balance calculations. The contract employs mathematical libraries to precisely compute prices and quantities along swap paths, incorporates secure transfer libraries to

prevent asset operation anomalies, and introduces signature-based authorization to enable liquidity removal without pre-approval, optimizing the trading experience. Overall, it implements liquidity management and efficient token exchange under an automated market maker (AMM) mechanism.

Tudst Contract

The PandaToken contract is an ERC20 standard-compliant token implementation designed to provide basic token management functionalities. It sets the token's name, symbol, decimals, and initial total supply through an initialization function in a single step, allocating all initial tokens to a designated owner address. The contract implements core ERC20 features, including token transfers, balance queries, allowance management, and transfer approval mechanisms, utilizing the SafeMath library to ensure arithmetic safety and prevent overflow or underflow. The transfer logic directly handles balance adjustments without incorporating complex mechanisms like fees, blacklists, or burning, though a predefined (but inactive) burn address is reserved for potential future extensions. The contract maintains a simple structure, omitting access control or advanced features, focusing on delivering a lightweight, standard-compliant token solution suitable for rapid deployment of basic tokens.

USDT Contract

This contract is a minimal token contract based on the OpenZeppelin ERC20 standard, primarily designed to create and issue a custom token named "2USDT." Inheriting the core logic of the ERC20 standard, the contract directly defines the token name and symbol as "2USDT" in the constructor and mints an initial supply of 10^40 (i.e., 10,000 trillion trillion) tokens to the contract creator's address upon deployment. The implementation relies entirely on the built-in functions of the ERC20 standard, without introducing additional token mechanisms (such as burning, fees, or permission management). It solely uses the _mint function for initial token distribution, resulting in highly concise code suitable for quickly deploying basic ERC20 tokens. Its core features

include standardized transfer/balance query functionality, a fixed total supply issuance model, and complete reliance on the audited OpenZeppelin contract library to ensure foundational security.

Usdtbsc Contract

The contract is a BEP20-compliant token implementation named "Tether USD" (USDT) with 18 decimal places. Its core functions include token transfers, balance queries, allowance management, and token minting/burning. Upon deployment, 30 million tokens are initially issued and fully allocated to the deployer's address. The contract owner holds exclusive minting rights and can issue additional tokens via the mint function, while any user can burn their own tokens using the burn function.

3.2 Vulnerability Classification Description

Likelihood \ Impact	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Info

3.3 Vulnerability Distribution

Severity	Found	Resolved	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	1	0	1
Info	1	0	1

4. Risk details

L-01 Use compiler version 0.8 or above, or employ SafeMath to avoid potential integer overflow

Impact: Medium

Likelihood: Low

Risk Level: Low

Details:

The Solidity compiler versions **0.8** and above include built-in integer overflow checks by default, or **SafeMath** can be used to detect integer overflows. The contract **WJU.sol** uses compiler version **0.6.0** and does not employ **SafeMath**, which may lead to potential integer overflow vulnerabilities. For example, in the following code, when wad is extremely large, it could cause an integer overflow in **balanceOf[dst]**.

```
function transferFrom(address src, address dst, uint wad)
    public
    returns (bool)
{
    require(balanceOf[src] >= wad);

    if (src != msg.sender && allowance[src][msg.sender] != uint(-1)) {
        require(allowance[src][msg.sender] >= wad);
        allowance[src][msg.sender] -= wad;
    }

    balanceOf[src] -= wad;
    balanceOf[dst] += wad;

    emit Transfer(src, dst, wad);

    return true;
}
```

Suggestion:

Use compiler version 0.8 or higher, or employ SafeMath to prevent integer overflows.

I-01 In the swap router, the functions handling token/native token swaps may pose a risk of unintended profit due to vulnerabilities

Impact: Low

Likelihood: Low

Risk Level: INFO

Details:

When the token's economic model is deflationary or employs a **Fee-on-Transfer (FoT)** token model, flawed implementation by developers may inadvertently allocate token incentives to the SwapRouter contract. In the swap logic, the SwapRouter contract should not hold any tokens. However, if tokens are erroneously deposited into the SwapRouter, attackers could exploit this by swapping tokens and native tokens. Specifically, functions like **removeLiquidityETH** or **removeLiquidityETHSupportingFeeOnTransferTokens** could be abused to drain the misallocated tokens from the SwapRouter.

```
function removeLiquidityETHSupportingFeeOnTransferTokens(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) public virtual override ensure(deadline) returns (uint amountETH) {
    (, amountETH) = removeLiquidity(
        token,
        WETH,
        liquidity,
        amountTokenMin,
        amountETHMin,
        address(this),
        deadline
    );
    TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
    IWETH(WETH).withdraw(amountETH);
    TransferHelper.safeTransferETH(to, amountETH);
}
```

```
function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
    (amountToken, amountETH) = removeLiquidity(
        token,
        WETH,
        liquidity,
        amountTokenMin,
        amountETHMin,
        address(this),
        deadline
    );
    TransferHelper.safeTransfer(token, to, amountToken);
    IWETH(WETH).withdraw(amountETH);
    TransferHelper.safeTransferETH(to, amountETH);
}
```

Examples of such vulnerabilities have been observed on Ethereum, such as the **FEI** Token incident.

Suggestions:

Advise developers to ensure that incentive tokens are not deposited into the SwapRouter when designing the economic model. Explicit safeguards should be implemented to prevent accidental token allocations to the router contract.

Reason:

If incentive tokens are mistakenly sent to the SwapRouter (e.g., due to flawed tokenomics or FoT logic), attackers could exploit functions like removeLiquidityETH to drain these tokens, leading to protocol losses.

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Lunaray Toolkit	self-developed toolkit

Disclaimer：

The Lunaray security team is only responsible for issuing reports and assuming corresponding responsibilities for facts that occurred or existed before the release of this report. Due to the inability to determine the security status of smart contracts for facts occurring after the release of this report, no responsibility is assumed for such matters.

The Lunaray security team conducts security audits on the security audit items in the project protocol and is not responsible for the project background and other circumstances. The subsequent on-chain deployment and operational methods of the project party are not within the scope of this audit.

This report is solely based on the information provided to Lunaray at the time of issuance for security auditing. If there is any concealment of project information or discrepancies between the reported situation and the actual circumstances, the Lunaray security team shall not be held responsible for any resulting losses or adverse impacts.

The market is risky, investment should be cautious. This report is only a security audit of the smart contract code and the announcement of the results, and does not constitute investment advice or basis.

https://lunaray.co

https://github.com/lunaraySec

https://x.com/lunaray_co

https://t.me/lunaraySec