SECURITY AUDIT OF

SWAPXV2

Public Report

Apr 15, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
BSC	Binance Smart Chain or BSC is an innovative solution for introducing interoperability and programmability on Binance Chain.
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.
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.
DEX	Decentralized exchanges (DEX), like PancakeSwap or Uniswap, are autonomous financial protocols powered by smart contracts that enable crypto traders to convert one digital asset for another with all transactions viewable on the blockchain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Apr 15, 2025. We would like to thank the UnicornX 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 SwapXV2. 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About SwapXV2

UnicornX is the fastest AI-driven on-chain trading ecosystem for Solana, combining secure multi-wallet management, real-time token discovery and swap execution, zero-latency market data, and advanced protections like anti-sandwich and transaction acceleration. With intuitive grouping and annotation of wallets, "many-to-many" fund transfers, automated profit reports, anomaly monitoring, and customizable alerts for both wallet and token activity, **UnicornX** empowers users to trade smarter and faster. Whether you're tracking your own portfolio or tapping into our social-trading network to mirror expert strategies and earn referral rewards, **UnicornX** delivers cutting-edge technology and ease of use in one seamless platform.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the SwapXV2. It was conducted on commit `29cbb2c58a24600b4491bff2583d44466db134c1` from git repository: *https://github.com/UnicornXAi/UnicornXSwapX/*

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`1044504efd9e17f5535b01e53b22236244c2186ac69ae08725669c9654a7654f`	`./SwapXV2.sol`

It was deployed on Binance Smart Chain (BSC) with address `0x9FCcbc0B42576a69Fe962F134fC908adf9D19502` and the following properties:

PROPERTY	VALUE
Proxy address	*https://bscscan.com/address/0xb804EbB99cDE3CA1B13e9173163D9ac409d13945*
Contract address	*https://bscscan.com/address/0x9fccbc0b42576a69fe962f134fc908adf9d19502*
Creator	`0xf3cdad212a8068c778008f1eb5606d35c74ffcb0`
Compiler version	v0.8.20+commit.a1b79de6

Table 1. SwapXV2's deploy properties

Note: This audit's scope is limited to the provided source code files. Some functions in the audited contracts are **upgradeable**, meaning the contract owner can modify the contract logic at any time in the future. Besides, the compiler version has some low-severity Solidity Compiler warnings.

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.

SEVERITY LEVEL	DESCRIPTION
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.4. Disclaimer

UnicornX 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. UnicornX understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, UnicornX 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 UnicornX will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the UnicornX, the final report will be considered fully accepted by the UnicornX without the signature.

2. AUDIT RESULT

2.1. Overview

The **SwapXV2** was written in `Solidity` language, with the required version to be `>=0.8.0`.

The contract extends upgradeable contracts through `PausableUpgradeable`, `ReentrancyGuardUpgradeable`, `IERC20Upgradeable`, `SafeERC20Upgradeable`, `OwnableUpgradeable`, and `IERC1271Upgradeable`. It also includes libraries for math functions and `Uniswap` operations.

SwapXV2 integrates multiple trading protocols: `Uniswap V2`, `Uniswap V3`, and `PancakeSwap V3`. This integration enables cross-protocol routing, where trades can flow through various protocol combinations (`V2-V2`, `V3-V3`, `V2-V3`, or`V3-V2`). Users benefit from better rates by accessing liquidity across the entire **DeFi** ecosystem instead of a single protocol.

The contract offers various swap types for different trading strategies. Users can choose between exact input swaps (specifying input amount with minimum output) and exact output swaps (specifying desired output with maximum input). For maximum efficiency, it supports both single-hop trades (direct token exchanges) and multi-hop trades (routes through intermediate tokens).

SwapXV2 handles token types efficiently. It manages both wrapped and unwrapped native tokens (`ETH/BNB`), automatically handling the wrapping process. The contract supports fee-on-transfer tokens and integrates with specialized contracts (`FourTokenManager` and `FlapTokenManager`) for specific token functionality.

The contract implements a fee system for trades, directing fees to a designated collector address. It maintains an exemption list for certain users or contracts that don't pay fees. The owner can adjust the fee rate to match market conditions.

2.2. Findings

#	Title	Severity	Status
1	Users can avoid the fee-in charge when using WETH	MEDIUM	ACKNOWLEDGED
2	`buyFlapToken` always reverts when called due to wrong fee calculation	LOW	ACKNOWLEDGED
3	User's amount can be stuck-in the Swap contract	LOW	ACKNOWLEDGED

2.2.1. MEDIUM - Users can avoid the fee-in charge when using WETH

Positions:

- SwapXV2.sol#swapV2MultiHopExactIn()
- SwapXV2.sol#swapV2MultiHopExactOut()

Description:

These two functions use **ETH** and **WETH** as the native `tokenIn`. However, they only mark `nativeIn` as "true" for **ETH**. If users set **WETH** as tokenIn, `nativeIn` will be false, allowing them to avoid paying fees for native tokens.

```
if (tokenIn == address(0)) {
        ...
            nativeIn = true;
            tokenIn = WETH;
        ...
```

RECOMMENDATION

The functions should verify both **ETH** and **WETH** values when applicable.

2.2.2. LOW - `buyFlapToken` might cause reverts when called due to wrong fee calculation

Positions:

- SwapXV2.sol#buyFlapToken()

Description:

The function sets amountIn to the total msg.value, then uses this amount to buy all tokens. Afterward, it calculates the fee from the realAmountIn.

```
uint256 amountIn = msg.value;

        uint256 bnbBalanceBefore = address(this).balance;

        amountOut = IFlapTokenManager(manager).buy{value: amountIn}(
            token,
            recipient,
            minAmount
        );
        uint256 bnbBalanceAfter = address(this).balance;
        uint256 realAmountIn = bnbBalanceBefore - bnbBalanceAfter;
        uint256 fee = takeFee(address(0), realAmountIn);
```

The problem is that the `takeFee` function uses a portion of `msg.value` to send fees to the `feeCollector`. Since all `msg.value` was already used to purchase tokens, this might cause the `takeFee` function to revert.

```
function takeFee(
        address tokenIn,
        uint256 amountIn
    ) internal returns (uint256) {
        if (feeExcludeList[msg.sender]) return 0;

        uint256 fee = amountIn.mul(feeRate).div(feeDenominator);

        if (tokenIn == address(0) || tokenIn == WETH) {
            require(address(this).balance > fee, "insufficient funds");
            (bool success, ) = address(feeCollector).call{value: fee}("");
            require(success, "SwapX: take fee error");
            emit FeeCollected(
                address(0),
                msg.sender,
                feeCollector,
                fee,
                block.timestamp
            );
        ...
```

RECOMMENDATION

The buying phase should not use the total `msg.value`.

2.2.3. LOW - User's amount can be stuck-in the Swap contract

Positions:

- SwapXV2.sol#buyMemeToken() - L252

Description:

The refund mechanism of `buyMemeToken` uses `funds - realAmountIn - fee` to calculate the refund amount. However, since funds is a user-provided parameter, if a user calls the function with a higher `msg.value` than funds, the excess amount might become stuck in the contract.

```
function buyMemeToken(
        address tokenManager,
        address token,
        address recipient,
        uint256 funds,
        uint256 minAmount,
        string memory memoStr
    ) external payable nonReentrant whenNotPaused returns (uint256 amountOut) {

        ...

        // refund
        (bool success, ) = address(msg.sender).call{
            value: funds - realAmountIn - fee
        }("");
        require(success, "refund bnb error");

        ...
    }
```

RECOMMENDATION

The refund mechanism should send back the total remaining amount to the user.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 15, 2025*	Public Report	Verichains Lab

Table 3. Report versions history