SECURITY AUDIT OF

AKRONSWAP SMART CONTRACTS

Public Report

Sep 19, 2024

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 Sep 19, 2024. We would like to thank the Akron Finance 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 Akronswap Smart Contracts. 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 identified some issues in the source code and provided recommendations. The Akron Finance team acknowledged these issues and gave clear explanations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Akronswap Smart Contracts

Akronswap is a decentralized exchange that protects LPs and swappers from negative maximal extractable value (MEV), specifically, LPs from loss-versus-rebalancing and swappers from sandwich attacks.

1.1.1. To passive liquidity providers on Akronswap

LPs, especailly passive LPs, capture a significant percentage of the arbitrage profits from arbitraguers from each of their swaps, through dynamic swap fees. In effect, LPs become "passive arbitrageurs": LPs on Akronswap stop losing money to arbitrageurs and start earning money together with arbitrageurs!

1.1.2. To swappers on Akronswap

Traders pay almost zero dynamic swap fees if their swap amount is small, because dynamic swap fee is at most the price impact (e.g. if the price impact is 0.5 bps, traders pay a fee of at most 0.5 bps).

Traders are protected from front running, back running and sandwich attacks, which generate huge profits for attackers each week.

Arbitrageurs can use Akronswap as "arbitrage opportunity of last resort" after opportunites on other pools have been arbitraged away.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Akronswap Smart Contracts. It was conducted on commit `1782fe28abe7d6f24d5ffa1fcb9bcaccb4c3adae` from git repository link *https://github.com/akron-finance/v2-core*.

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

SHA256 Sum	File
`f1716b4a6032cd4e6d07a407a358d50ac670c52a5493ac776ab1930d1b71c452`	`./UniswapV2Factory.sol`
`2f4052a4517dea5ab40c1f34a4ae8535155dae4094a744ab57909843dc9b7e4e`	`./UniswapV2ERC20.sol`
`026c41ea2c85be5c3e2df76c819c9127ed5f718f4ce78783aa0abe030d4bdc9e`	`./SushiRoll.sol`

91a15af18ba3036e345f3646618c79480f481e67490b74e9ab9d8eb40cc4bfba	./libraries/UniswapV2OracleLibrary.sol
687c7bd261db3ecaf6de5c19a66c2c5dc06614ca24224e23551d497e9092775c	./libraries/UniswapV2Library.sol
c3e07a9975e7c206617aba8fe8ba63059eaa2bdfa504a8cf1bd461527f5fb3f4	./libraries/SafeMath.sol
24283a562d299a5e4133d3f05304eec8e75a1b18c6907dd2a8f399eea0b16524	./libraries/UQ112x112.sol
a553dd23aa798c18e1b2a19b2f64a2ba8144df56e212f20bab346be5c37287bb	./libraries/Math.sol
68f11e1cf4e19227867e3bed7a149bf1e366b4ea8fb0d5753fa1fc7450c76c54	./libraries/UniswapV2LiquidityMathLibrary.sol
26efb144f8468217af5043881c24a699eebb20a7b39a7811a53789489c526246	./UniswapV2Router01.sol
a2bcca715853f3b43b451147456e0a9e757193eb56a04fc6fad165d02dc2a409	./UniswapV2Router02.sol
9a33bcf0ac40e25ce462ce07c6cd9118c500b833aef4e10d400ea328591f653a	./UniswapV2Pair.sol

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 1. Severity levels

1.4. Disclaimer

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

2. AUDIT RESULT

2.1. Overview

The Akronswap Smart Contracts is a forked version of Uniswap V2 and SushiSwap, with some important modifications:

The code for the AMM constraint check in the `UniswapV2Pair` contract has been updated to a stricter formula. The original constraint from Uniswap V2 (excluding fees) can be expressed as follows:

$$final\_balance0 * final\_balance1 >= before\_swap\_balance1 * before\_swap\_balance0$$

And this constraint has been replaced with a new one, which can be expressed as follows:

$$(increase\ in\ balance0) * (final\_balance1) \geq (decrease\ in\ balance1) * final\_balance0$$

The new formula is stricter than the original, requiring the execution price of the swapper to be higher than the pool's price after the swap (for purchases, denominated in the input token).

To prove the correctness of the new formula, we must prove that the old constraint is still satisfied if the new constraint holds. Assuming the input token is `token0` and the output token is `token1`, the proof is as follows:

Let's define the following variables:

$$\begin{aligned} b_0 &: before\_swap\_balance0 \\ b_1 &: before\_swap\_balance1 \\ f_0 &: final\_balance0 \\ f_1 &: final\_balance1 \end{aligned}$$

The new AMM constraint can be expressed as follows:

$$\begin{aligned} & (increase\ in\ balance0) * (final\_balance1) \geq (decrease\ in\ balance1) * final\_balance0 \\ \Leftrightarrow\ & (f_0 - b_0)f_1 \geq (b_1 - f_1)f_0 \\ \Leftrightarrow\ & f_0 f_1 - b_0 f_1 \geq b_1 f_0 - f_1 f_0 \qquad (1) \end{aligned}$$

And we also have the following inequality:

$$\begin{aligned} & (b_1 - f_1)(f_0 - b_0) \geq 0 \\ \Leftrightarrow\ & b_1 f_0 - f_1 f_0 \geq b_1 b_0 - f_1 b_0 \qquad (2) \end{aligned}$$

Combine (1) and (2), we have:

$$
\begin{aligned}
& f_0 f_1 - b_0 f_1 \geq b_1 b_0 - f_1 b_0 \\
\Leftrightarrow\ & f_0 f_1 \geq b_1 b_0 \\
\Leftrightarrow\ & final\_balance0 * final\_balance1 >= before\_swap\_balance1 * before\_swap\_balance0
\end{aligned}
$$

The final formula is identical to the original constraint in Uniswap V2. Therefore, we can conclude that any swap that satisfies the new constraint also satisfies the original Uniswap V2 constraint.

2.2. Findings

During the audit process, the audit team found some issues and recommendations in the given version of Akronswap Smart Contracts.

#	Issue	Severity	Status
1	Preventing same-direction trades in a block can negatively impact user experience	MEDIUM	ACK
2	Unclear of invariant formula after changes	INFORMATIVE	ACK
3	Unclear changes of the `getAmountOut` formula	INFORMATIVE	ACK

2.2.1. Preventing same-direction trades in a block can negatively impact user experience MEDIUM

Affected files:

- v2-core/contracts/UniswapV2Pair.sol

The idea of preventing same-direction trades within a single block was originally intended to reduce the risk of front-running, back-running, and sandwich attacks. However, implementing this mechanism would drastically reduce both trading volume and user experience. For instance, if 10 users submit trades during an active market period and only one trade is successfully executed, the other 9 users would receive transaction reversion errors, which can be extremely frustrating for them.

To make matters worse, this mechanism could prevent the AMM platform from being integrated with other DeFi protocols that require multiple trades to be executed within a single block.

```
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external
lock {
    // ...
    uint blockNumber = block.number;
    if (_reserve0 > balance0) {
        if (oneForZeroBlockNumberLast == blockNumber) revert('SWAP_NOT_ALLOWED');
        oneForZeroBlockNumberLast = blockNumber;
    }
    if (_reserve1 > balance1) {
        if (zeroForOneBlockNumberLast == blockNumber) revert('SWAP_NOT_ALLOWED');
        zeroForOneBlockNumberLast = blockNumber;
    }
    // ...
}
```

UPDATES

- **Sep 19, 2024**: The Akron Finance team acknowledged the issue and provided an explanation. They expect aggregators and arbitrageurs to be the main sources of swap flows, and they will move to the next best pool when reverts occur. As a result, a few aggregators will compete to secure the only swap in the block.

2.2.2. Unclear of invariant formula after changes INFORMATIVE

Affected files:

- v2-core/contracts/UniswapV2Pair.sol

As we can see from the description provided by the dev team, the code for the invariant check has been changed. The original version of Uniswap V2 used the following formula, which ensured that the product of the two reserves remained greater than or equal to the previous value:

```
require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
{ // scope for reserve{0,1}Adjusted, avoids stack too deep errors
    uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
    uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
    require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
}
```

This has been updated to:

```
require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
{ // scope for reserve{0,1}Adjusted, avoids stack too deep errors
    if (
        (balance0 > _reserve0 ? balance0 - _reserve0 : 0).mul(balance1)
            < (_reserve1 > balance1 ? _reserve1 - balance1 : 0).mul(balance0)
            || (balance1 > _reserve1 ? balance1 - _reserve1 : 0).mul(balance0)
            < (_reserve0 > balance0 ? _reserve0 - balance0 : 0).mul(balance1)
    ) revert('K');
}
```

Changing the core AMM formula from the constant product `x * y = K` requires a clear explanation of the new formula and its implications. Please provide us with the mathematical formula for the new curve so that we can ensure the invariant still holds.

UPDATES

- **Sep 19, 2024**: The Akron Finance team acknowledged the issue and provided a detailed explanation of the new formula. They also provided a proof that the invariant still holds under the new formula.

2.2.3. Unclear changes of the `getAmountOut` formula INFORMATIVE

Affected files:

- v2-core/contracts/libraries/UniswapV2Library.sol

The `getAmountOut` function in the `UniswapV2Library` contract has been updated. The original formula was:

```
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns
(uint amountOut) {
    require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(997);
    uint numerator = amountInWithFee.mul(reserveOut);
    uint denominator = reserveIn.mul(1000).add(amountInWithFee);
    amountOut = numerator / denominator;
}
```

This is replaced with the new formula:

```
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns
(uint amountOut) {
    require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint numerator = reserveOut.mul(amountIn);
    uint denominator = amountIn.mul(2).add(reserveIn); // AUDIT: UNCLEAR FORMULA
    amountOut = numerator / denominator;
}
```

The original version of the formula can be rewritten as follows (if we ignore the fee):

```
amountOut = amountIn * reserveOut / (reserveIn + amountIn)
```

However, the new formula is as follows:

```
amountOut = amountIn * reserveOut / (reserveIn + amountIn * 2)
```

Can you provide us with a clear explanation for why the `amountIn` in the new formula is multiplied by `2`? With this new formula, the `amountOut` of tokens that a user receives when swapping using the Router contract will be significantly lower than with the original formula, which seems counterintuitive.

UPDATES

- **Sep 19, 2024**: The Akron Finance team acknowledged the issue and provided a detailed explanation of the new formula. From the proof of the modified AMM constraint, we can derive the updated formula for `getAmountOut` as shown above.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 19, 2024*	Public Report	Verichains Lab

Table 2. Report versions history