SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

DEXXROUTER

Overview

Project Summary

- Name: DEXXRouter
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	DEXXRouter
Version	v1
Type	Solidity
Dates	Oct 03 2024
Logs	Oct 03 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	1
Total	3

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	There is no check to verify if the pair is correct	High	Risky External Calls	Pending
2	The feeBasisPoints has no upper limit	Low	Data Validation	Pending
3	Use of floating pragma	Informational	Configuration	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. There is no check to verify if the pair is correct

Severity: High	Category: Risky External Calls
Target: - DEXXRouter.sol	

Description

The `BuyTokenWithETH()` and `SellTokenToETH()` functions only accept the `_pair` parameter, but do not verify whether the pair corresponds to the expected two tokens.

DEXXRouter.sol:L304-L321

```
function BuyTokenWithETH(address _pair,address _tokenOut,uint _amountOutMin,uint256
_tip,bool _v3) external payable returns (uint amountOut) {
    uint fee = msg.value * getTokenSwapFee(_tokenOut) / 1000;
    uint swapValue = msg.value - fee;
    if ( _tip > 0 ) {
        swapValue = msg.value - _tip;
    }
    uint _preTokenBalance = IERC20(_tokenOut).balanceOf(msg.sender);
    if (_v3) {
        _v3Buy(_pair,WETH,_tokenOut,swapValue,msg.sender);
    } else {
        _v2Buy(_pair,WETH, swapValue,msg.sender);
    }
    ...
}
function _v3Buy(address _pair,address _tokenIn,address _tokenOut,uint _amountIn,address
_receipt) internal {
    currentPool = _pair;
    bool zeroForOne = _tokenIn < _tokenOut;
    IPoolV3(_pair).swap(_receipt, zeroForOne, int256(_amountIn), zeroForOne ?
Constant.MIN_SQRT_RATIO + 1 : Constant.MAX_SQRT_RATIO - 1,
abi.encode(_tokenIn,_receipt));
}
```

If an attacker writes a malicious pair contract, they can use the `BuyTokenWithETH()` or `SellTokenToETH()` functions in the router contract to have the router contract actively call the attacker's contract and set it as the `currentPool`. In this case, the attacker's contract can directly call `uniswapV3SwapCallback()`, allowing it to withdraw all the Ether from the contract or any tokens that other users have approved for this contract.

Recommendation

Consider checking if the pair is a valid Uniswap V3 pool.

2. The feeBasisPoints has no upper limit

Severity: Low	Category: Data Validation
Target: - DEXXRouter.sol	

Description

The `feeBasisPoints` variable in the contract does not have an upper limit for modifications. If `feeBasisPoints` is set too high, it poses a risk to users.

DEXXRouter.sol:L142-L144

```
function changeFee(uint256 _newFee) public onlyOwner {
    feeBasisPoints = _newFee;
}
```

Recommendation

It is recommended to add an upper limit for the `feeBasisPoints` variable.

2.3 Informational Findings

3. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - DEXXRouter.sol	

Description

```
pragma solidity ^0.8.0;
```

The DEXXRouter uses a floating compiler version `^0.8.0`.

Using a floating pragma `^0.8.0` statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
DEXXRouter.sol	06b5bc30ff9cfca088d04cbbcc9fd64759516dd1