Kekotron Security Review

Pashov Audit Group

Conducted by: Dan Ogurtsov, T1MOH, tsvetanovv

January 16th 2024 - January 21st 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **KekotronRouter** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Kekotron Router

The router is a general purpose swapping router designed to handle Uniswap v2 and v3 style swaps. It has dynamic handling to ensure safety and consistency for all supported protocols.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

***review commit hash* - 1787678a9dc5bb8fb1879f695f1c09455c5ff34e**

***fixes review commit hash* -**

Scope

The following smart contracts were in scope of the audit:

- `KekotronRouterV2`
- `KekotronRouterInitializer`
- `KekotronSwapV2`
- `KekotronSwapTraderJoeV2_1`
- `Constants`
- `DataTypes`
- `KekotronErrors`
- `KekotronSwapV3`
- `KekotronLib`
- `interfaces/**`

7. Executive Summary

Over the course of the security review, Dan Ogurtsov, T1MOH, tsvetanovv engaged with Kekotron Terminal to review Kekotron Router. In this period of time a total of **11** issues were uncovered.

Protocol Summary

Protocol Name	Kekotron Router
Repository	https://github.com/astralparrot/KekotronRouter/
Date	January 16th 2024 - January 21st 2024
Protocol Type	Multi-chain, multi-factory router

Findings Count

Severity	**Amount**
High	1
Medium	3
Low	7
Total Findings	**11**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Users can set feeOn > 1 to avoid fees	High	Resolved
[M-01]	Choosing FeeIn is always more beneficial than FeeOut	Medium	Resolved
[M-02]	Not checked ETH provided == amountIn swapped	Medium	Resolved
[M-03]	Missing deadline check in swap functions	Medium	Resolved
[L-01]	allowedV2Callbacks - no way to edit	Low	Resolved
[L-02]	safeTransferETH() may revert due to a hardcoded gas limit	Low	Resolved
[L-03]	Confusing naming for chains with non-ETH native currency	Low	Resolved
[L-04]	Minimal sqrtPriceLimitX96 slightly differs from actual minimal price limit	Low	Resolved
[L-05]	Swap unexpectedly executes when tokenIn == tokenOut	Low	Resolved
[L-06]	KekotronSwapV3.sol doesn't support pools with fee > 6.5535%	Low	Resolved
[L-07]	safeTransfer() and safeTransferFrom doesn't check contract existence	Low	Resolved

8. Findings

8.1. High Findings

[H-01] Users can set feeOn > 1 to avoid fees

Severity

Impact: Medium, less fee received for the protocol

Likelihood: High, a simple input change on every swap

Description

`feeOn` is indicated by users on every swap and is responsible for choosing between `feeIn` and `feeOut`. `feeOn` == 0, means `feeIn` true and `feeOut` false. `feeOn` == 1, means `feeIn` false and `feeOut` true.

```
(bool feeIn, bool feeOut) = fee > 0 ? (feeOn == 0, feeOn == 1) : (false, false);
```

But if `feeOn` is above 1, both `feeIn` and `feeOut` will be false, even if `fee` is set. As a result, fees will not be accrued.

Recommendations

Consider checking that `feeOn` is either 0 or 1, or just use boolean for feeOn input (given that in tests it is always converted between boolean and uint8)

8.2. Medium Findings

[M-01] Choosing FeeIn is always more beneficial than FeeOut

Severity

Impact: Low, the difference is very small for the majority of swaps

Likelihood: High, FeeIn can be chosen on every swap

Description

Users can choose between `FeeIn` (fee on tokenIn-amountIn), or choose `FeeOut` (fee on tokenOut-amountOut). If both alternatives are calculated it always happens that choosing `FeeIn` always results in fewer fees (the user receives more net amountOut).

Technically it happens because FeeIn means less funds to swap, and less loss due to slippage in the end. The difference between the two options is becoming larger when the size of the swap grows (more slippage).

The difference between options is 0.5% for a swap of 100% of reserves, and 0.09% for 10% of reserves.

But, `FeeIn` is also more beneficial for FeeReceiver as the fee is taken before slippage and DEX fees.

As a result, `FeeIn` is always financially better for both the protocol and the user.

Recommendations

Consider some of the options:

1. leaving only FeeIn or only FeeOut as a default setup
2. adjust the formula so that the two options are identical
3. leaving as it is (and acknowledging the difference as neglectable)

[M-02] Not checked ETH provided == amountIn swapped

Severity

Impact: High, loss of funds

Likelihood: Low, required mistake in inputted data and choosing native token as tokenIn

Description

RouterV2 does not check that `msg.value` is equal to `amountIn` in case of `tokenIn == address(0)`. If a user sends in fact less - the transaction will revert. If a user sends in fact more - some native token will stuck on Router after the swap (the delta will not be swapped)

Recommendations

Consider checking that `msg.value == amountIn` if the native token is `tokenIn`.

[M-03] Missing deadline check in swap functions

Severity

Impact: Low, the user would not be at a loss and would only miss positive slippage.

Likelihood: High, deadline parameter is missing.

Description

Swap functions don't have deadline parameter. This parameter can provide the user an option to limit the execution of their pending transaction. Without a deadline parameter, users can execute their transactions at unexpected times when market conditions are unfavorable.

However, this is not a big problem in this case because the functions have slippage protection. Even though the users will get at least as much as they set, they may still be missing out on positive slippage if the exchange rate becomes favorable when the transaction is included in a block.

Recommendations

Introduce a `deadline` parameter in all swap functions.

8.3. Low Findings

[L-01] allowedV2Callbacks - no way to edit

RouterV2 has to mappings for selected callbacks:

```
/// @dev Fallback selectors for callbacks, allowing for adding additional
// protocols in the future
    mapping(bytes4 => bool) public allowedV2Callbacks;
    mapping(bytes4 => bool) public allowedV3Callbacks;
```

But only V3 can be edited with no function for V2.

```
function updatedAllowedV3Callbacks
      (bytes4 selector, bool allowed) external onlyOwner {
        allowedV3Callbacks[selector] = allowed;

        emit AllowedV3CallbackUpdated(selector, allowed);
    }
```

Consider a function to make changes in `allowedV2Callbacks` too.

[L-02] `safeTransferETH()` may revert due to a hardcoded gas limit

In `KekotronLib.sol` we have `safeTransferETH()` which is used for ETH transfer:

```
function safeTransferETH(address to, uint256 value) internal {
        (bool success,) = to.call{value: value, gas: PUSH_PAYMENT_GAS_LIMIT}
          (new bytes(0));
        if (!success) {
            revert EthTransfer();
        }
    }
```

The function is used in many places in the protocol when sending ETH. The problem is that the function uses a fixed value of gas:

```
uint256 constant PUSH_PAYMENT_GAS_LIMIT = 10_000;
```

It is not good to have the gas limit fixed. This will cause the function to revert if insufficient gas is used. Even in the solidity documentation, we can read:

> It is best to avoid relying on hardcoded gas values in your smart contract code, regardless of whether state is read from or written to, as this can have many pitfalls. Also, access to gas might change in the future.

Consider removing the gas field or increase the gas limit.

[L-03] Confusing naming for chains with non-ETH native currency

In code you use function `_weth()` to get address of wrapped ERC20 of native currency. Also functions like `_swapExactEthForTokensV_X()` and `_swapExactTokensForEthV_X()`.

Problem is that protocol is supposed to deploy on different chains including BSC, Polygon, Avalanche. They have non-ETH native currencies and functions to swap from/into native currency won't work on them if naively set address of WETH to `weth` variable instead of Wrapped Native token.

Recommendation: replace wording of weth and eth to native currency.

[L-04] Minimal `sqrtPriceLimitX96` slightly differs from actual minimal price limit

Currently value 4295128749 is used:

```
uint160 sqrtPriceLimitX96 =
  (zeroForOne ? 4295128749 : 1461446703485210103287273052203988822378723970341);
```

However the smallest value is `4295128739 + 1` according to UniswapV3:

```
uint160 internal constant MIN_SQRT_RATIO = 4295128739;
```

As a result, trade can't occur at the extremely low price.

[L-05] Swap unexpectedly executes when `tokenIn == tokenOut`

Currently if user submits `tokenIn == tokenOut`, router will trade tokenOut for tokenIn despite calldata is invalid and therefore trade should not occur.

Add additional checks:

```
function _swapV2(
      Swapmemoryparam,
      addressto,
      uint16poolFee
    ) private returns (uint256
+       require(param.tokenIn != param.tokenOut);
        bool zeroForOne = param.tokenIn < param.tokenOut;

        ...
    }

    function _deriveData
      (Swap memory param, address payer, uint16 poolFee, uint8 protocol)
        private
        pure
        returns (bool, int256, uint160, bytes memory)
    {
+       require(param.tokenIn != param.tokenOut);
        bool zeroForOne = param.tokenIn < param.tokenOut;

        ...
    }

    function _swapV2_1(Swap memory param, address to) private returns
      (uint256) {
+       require(param.tokenIn != param.tokenOut);
        bool zeroForOne = param.tokenIn < param.tokenOut;

        ...
    }
```

[L-06] KekotronSwapV3.sol doesn't support pools with fee > 6.5535%

Kekotron stores poolFee as uint16, max value is 65535 which means fee is 6.5535%. To handle pools with higher fee, it shifts `poolFee` by 10 into uint24. However computation is performed in uint16 type, therefore it will revert in case result is higher than 65335.

```
function _deriveData
      (Swap memory param, address payer, uint16 poolFee, uint8 protocol)
        private
        pure
        returns (bool, int256, uint160, bytes memory)
    {
        bool zeroForOne = param.tokenIn < param.tokenOut;
        // We shift the fee to save some calldata space in the original call
@>      uint24 shiftedFee = poolFee * 10;

        ...
    }
```

Here is PoC

Recommendation: cast to uint24

```
-       uint24 shiftedFee = poolFee * 10;
+       uint24 shiftedFee = uint24(poolFee) * 10;
```

[L-07] `safeTransfer()` and `safeTransferFrom` doesn't check contract existence

Current implementation will succeed if `token` address is not smart contract, because all calls to addresses with zero code return `success == true`.

```
function safeTransfer(address token, address to, uint256 value) internal {
@>      (bool success, bytes memory data) = token.call(abi.encodeWithSelector
  (IERC20.transfer.selector, to, value));
        if (!(success && (data.length == 0 || abi.decode(data, (bool))))) {
            revert TokenTransfer();
        }
    }

    function safeTransferFrom
      (address token, address from, address to, uint256 value) internal {
        (bool success, bytes memory data) =
@>          token.call(abi.encodeWithSelector
  (IERC20.transferFrom.selector, from, to, value));
        if (!(success && (data.length == 0 || abi.decode(data, (bool))))) {
            revert TokenTransferFrom();
        }
    }
```

This behaviour can be overlooked in future, which results in potential issue.

Add sanity check as best practice:

```
+       require(address.code.length > 0);
```