0xCommit

Security Assessment Report

KASU Finance

KASU contracts Audit

Version: Final

Date: 12 Jun 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

Introduction

Purpose of this report

0xCommit has been engaged by **KSU Finance** to perform a security audit of several Token and Vault contract components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following commits:

Github Link: https://github.com/Kasu-Finance/kasu-contracts.git

Version	Commit hash
Initial	5e2b70d3b90172a1422192b1aa95522a01bf1e2b

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
High	An attacker can successfully execute an attack that clearly results in operational issues for the service. This also includes any value loss of unclaimed funds permanently or temporary.
Medium	The service may be susceptible to an attacker carrying out an unintentional action, which could potentially disrupt its operation. Nonetheless, certain limitations exist that make it difficult for the attack to be successful.
Low	The service may be vulnerable to an attacker executing an unintended action, but the impact of the action is negligible or the likelihood of the attack succeeding is very low and there is no loss of value.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
    d. Access Control Issues
    e. Boundary Analysis
4. Report preparation

Functionality Overview

Summary of Findings

Sr. No.	Description	Severity	Status
1	Swapper flow does not handle native token returns.	Low	Resolved
2	Previous balance not checked leading to accounting issues.	Low	Resolved

Detailed Findings

1. Swapper flow does not handle native token returns.

Severity: Low

Description

The Swapper Contracts operate under the assumption that all swap operations involve inherent token transfers. However, certain swap operations, such as "**swapTokensForExactETH**" and "**swapExactETHForTokens**" in the Uniswap V2 Router, return native tokens. (There can be other swap operations which return native tokens.) This specific scenario is not accounted for in the current code, leading to the disruption of normal operations for certain users, resulting in transaction reversion. The issue stems from the fact that the swap function in swapper.sol does not have a payable modifier and it fails to handle cases where native tokens are returned from swap operations.

Remediation

- The swap function in swapper.sol should be updated to include the payable modifier.
- If tokenOut equals WETH token, implement pre- and post-swap checks on the native token balance to detect if any swap operation returns native tokens. If native tokens are received, convert the ETH in the swapper to WETH tokens and proceed with the transfer as usual.

Following is the remediated code for swap function -

Unset

```
// Swapper.sol - Remediated code

          function swap(address[] calldata tokensIn, SwapInfo[] calldata swapInfo, address
tokenOut, address receiver)
              external
              payable
              onlyRole(ROLE_SWAPPER, msg.sender)
              returns (uint256 tokenAmount)
          {
              if (swapInfo.length == 0 || tokensIn.length == 0 || tokenOut == address(0) ||
receiver == address(0)) {
                  revert InvalidSwapData();
              }

              uint256[] memory amountsIn = new uint256[](tokensIn.length);

              for (uint256 i; i < tokensIn.length; ++i) {
```

```
            amountsIn[i] = IERC20(tokensIn[i]).balanceOf(address(this));
        }

        // Perform the swaps.
        for (uint256 i; i < swapInfo.length; ++i) {
            if (!_exchangeAllowlist[swapInfo[i].swapTarget]) {
                revert ExchangeNotAllowed(swapInfo[i].swapTarget);
            }
            _approveMax(IERC20(swapInfo[i].token), swapInfo[i].swapTarget);

            // UPDATED CODE
            uint256 preswap = address(this).balance;
            swapInfo[i].swapTarget.functionCall(swapInfo[i].swapCallData);
            uint256 postSwap = address(this).balance;

            if ((preswap < postSwap) && (address(tokenOut) == address(_weth))){
                _weth.deposit{value : postSwap}();
            }
            // UPDATED CODE

            _resetApproval(IERC20(swapInfo[i].token), swapInfo[i].swapTarget);
        }

        tokenAmount = IERC20(tokenOut).balanceOf(address(this));
        if (tokenAmount > 0) {
            IERC20(tokenOut).safeTransfer(receiver, tokenAmount);
        }

        // Return remaining tokens.
        for (uint256 i; i < tokensIn.length; ++i) {
            uint256 tokenInBalance = IERC20(tokensIn[i]).balanceOf(address(this));
            if (tokenInBalance > 0) {
                IERC20(tokensIn[i]).safeTransfer(receiver, tokenInBalance);
            }
        }

        // Send ETH outside

        emit Swapped(receiver, tokensIn, tokenOut, amountsIn, tokenAmount);
    }
```

References

- https://docs.uniswap.org/contracts/v2/reference/smart-contracts/router-01#swapexacttokensforeth
- https://docs.uniswap.org/contracts/v2/reference/smart-contracts/router-01#swapethforexacttokens
- https://docs.uniswap.org/contracts/v2/reference/smart-contracts/router-01#swapethforexacttokens
- https://docs.uniswap.org/contracts/v2/reference/smart-contracts/router-02#swaptokensforexacteth

Status

Resolved

2. Previous balance not checked leading to accounting issues.

Severity: Low

Description

The function _transferAndSwap in DepositandSwap.sol transfers incoming tokens directly to swapper.sol without first verifying the balance of that token in swapper. This oversight can result in accounting discrepancies, as users may either intentionally or inadvertently transfer tokens directly to the swapper, and these funds could then be utilized by other users during their swapping operations. This situation could lead to unfair accounting advantages for some users at the expense of others' mistakes.

This behavior may not directly affect the protocol but hackers can misuse this issue in creative ways. (i.e. - Used in supply chain attacks)

Remediation

Prior to transferring tokens to the swapper for swap operations, it's crucial to verify the balance of tokenIn in the swapper. If a balance exists, instruct the swapper to transfer the balance of tokenIn to a recovery wallet defined by the admin. This proactive measure ensures that any existing token balances are safeguarded before initiating further swap operations.

Following is the remediated code -

Unset

```
// DepositSwap.sol - Remediated code
          /* ========== STATE VARIABLES ========== */

           /// @dev WETH contract.
           IWETH9 private immutable _weth;
           /// @dev Swapper contract.
           ISwapper private immutable _swapper;

           // UPDATE DONE
           address private _rescuer;
           // UPDATE DONE

           /* ========== CONSTRUCTOR ========== */

           /**
            * @notice Constructor.
            * @param weth_ WETH contract.
            * @param swapper_ Swapper contract.
            */
           constructor(IWETH9 weth_, ISwapper swapper_ , address rescuer_) {
               AddressLib.checkIfZero(address(weth_));
```

```
                AddressLib.checkIfZero(address(swapper_));

                _weth = weth_;
                _swapper = swapper_;
                _rescuer = rescuer_;
            }

            /* ========== INTERNAL MUTATIVE FUNCTIONS ========== */

            /**
             * @notice Swaps tokens using the specified swap information.
             * @dev Tokens are transferred from the caller to the swapper.
             * swapper should swap tokens and return the swapped tokens to this contract.
             * @param swapDepositBag Swap and deposit information.
             * @param outToken Expected token to receive.
             * @return outTokenAmount Amount of the token received.
             */
            function _transferAndSwap(SwapDepositBag memory swapDepositBag, address outToken)
                internal
                returns (uint256 outTokenAmount)
            {
                if (swapDepositBag.inTokens.length != swapDepositBag.inAmounts.length) revert
InvalidArrayLength();
                uint256 msgValue = msg.value;

                // Wrap ETH if any.
                if (msg.value > 0) {
                    _weth.deposit{value: msgValue}();
                }

                // Transfer the tokens from the caller to the swapper.
                for (uint256 i; i < swapDepositBag.inTokens.length; ++i) {
                // UPDATE DONE
                    uint256 swapperbalance =
IERC20(swapDepositBag.inTokens[i]).balanceOf(address(_swapper));
                    if ( swapperbalance > 0 ){
                        _swapper.rescueFund(swapDepositBag.inTokens[i] , _rescuer ,
swapperbalance );
                    }
                // UPDATE DONE

                    IERC20(swapDepositBag.inTokens[i]).safeTransferFrom(
                        msg.sender, address(_swapper), swapDepositBag.inAmounts[i]
                    );

                    if (swapDepositBag.inTokens[i] == address(_weth) && msgValue > 0) {
                        IERC20(address(_weth)).safeTransfer(address(_swapper), msgValue);
                    }
                }

                _swapper.swap(swapDepositBag.inTokens, swapDepositBag.swapInfo, outToken,
address(this));

                outTokenAmount = IERC20(outToken).balanceOf(address(this));

                if (outTokenAmount < swapDepositBag.minAmountOut) {
```

```
                    revert InsufficientOutputAmount(outTokenAmount,
swapDepositBag.minAmountOut);
                }
            }


//Changes in Swapper.sol - added rescueFund function.

      function rescueFund( address _token , address _rescuer, uint256 _amt )
            external
            onlyRole(ROLE_SWAPPER, msg.sender)
            returns ( bool )
      {
            return IERC20( _token ).transfer( _rescuer , _amt );
      }
```

Status

Resolved