SHERLOCK

SHERLOCK SECURITY REVIEW FOR

SHERLOCK

Prepared for:	dodo
Prepared by:	Sherlock
Lead Security Expert:	ak1
Dates Audited:	November 9 - November 12, 2022
Prepared on:	November 23, 2022

Introduction

DODO is a decentralized trading platform that uses the innovative Proactive Market Maker (PMM) algorithm to provide efficient on-chain liquidity for Web3 assets.

Scope

The following contracts in the DODOEX/dodo-route-contract repo are in scope.

- /SmartRoute/DODORouteProxy.sol
- DODOApprove.sol
- DODOApproveProxy.sol
- IDODOApprove.sol
- lib/DecimalMath.sol
- lib/UniversalERC20.sol

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
11	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

ctf_sec
jayphbee
TrungOre
ak1
Tomo
0xNazgul
0x52
__141345__
yixxas

0x4non
sach1r0
Nyx
Bnke0x0
zimu

simon135
koxuan
pashov
8olidity
rvierdiiev

ElKu
virtualfact
defsec

Issue M-1: Possible DOS inside the mixSwap and _multiSwap when for loop iteration goes long

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/82

Found by

0xNazgul, ak1

Summary

When look at the mixSwap and _multiSwap, for swap, for loop is used to go through for multiple pairs. When number of pair is more, the loop iteration will go long and this could lead to potential DOS.

Vulnerability Detail

in external swap,

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L278-L293

in _multiSwap, https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L393-L441

For each iteration, there function calling, decoding, transferring the fund is happended.

The number of iteration and work involved will be huge for _multiSwap

when there are more number of traversal, the contract can not work and revert due to out of gas.

Impact

The contract can not function and revert due to out of gas.

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L393-L441

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L278-L293

Tool used

Manual Review

Recommendation

Put ca on the number of pair or splitNumber for iteration. Split the swap data and process in separate transaction.

Issue M-2: DODORouteProxy.sol#L170: Lack of slippage protection for `externalSwap`

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/77

Found by

__141345__, jayphbee, ak1

Summary

When calling the `externalSwap` function, `minReturnAmount` is used to protect from unfair slippage events. But there is no validation whether the `minReturnAmount>0`.

I am raising this as issue by looking at other places where this validation is done.

for `mixswap`, the check is done here, https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L254

for `dodoMutliSwap`, the check is done here, https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L339

Vulnerability Detail

For `externalSwap`, there is no check whether the `minReturnAmount` value is greater than zero or not.

but , other type swaps has this validation. Refer the summary section for code reference where other functions has this check.

Impact

There will not be any protection from slippage if the `minReturnAmount` is not validated.

The check from `routewithdraw` will not catch this flaw.

```
require(receiveAmount >= minReturnAmount, "DODORouteProxy: Return amount is not
↪  enough");
```

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L164-L179

Tool used

Manual Review

Recommendation

Check whether the `minReturnAmount>0` inside the `externalSwap` function.

or

Add this check inside the `_routeWithdraw` function so that all other swaps will be covered.

Discussion

Evert0x

Low severity non-zero check

Issue M-3: calldata is not validated meaning an attacker can call arbitrary calldata to make reentrancy or steal funds

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/55

Found by

zimu, simon135, ctf_sec, koxuan, Bnke0x0, ak1

Summary

Calldata is not validated meaning an attacker can call arbitrary calldata to make reentrancy or steal funds/get out of fees . When calldata bytes are not validated you can do arbitrary operations that shouldn't be done.

Vulnerability Detail

When we are swaping tokens The attacker calls swapTarget with arbitrary `callDataConcat` which then they can use to reenter or do another operation with that calldata to call some contract. From a user perspective if calldata is not validated then a user can call a swap target that can't handle the `callDataConcat` and the function will revert.

Impact

An attacker can swap a lot more in the swap target than the function knows and then the attacker doesn't have to pay as much in fees steps: attacker transfers 5 eth but in the swapTarget they swap 6 ether then they get out of paying more fees but also if they swap less then users can loose funds and pay to much fees

Code Snippet

```
{
            require(swapTarget != _DODO_APPROVE_PROXY_, "DODORouteProxy: Risk
↪  Target");
            (bool success, bytes memory result) = swapTarget.call{
                value: fromToken == _ETH_ADDRESS_ ? fromTokenAmount : 0
            }(callDataConcat);
            // revert with lowlevel info
            if (success == false) {
                assembly {
                    revert(add(result,32),mload(result))
                }
            }
```

```
        }
```

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L205

Tool used

Manual Review

Recommendation

decode the data and go through some sort of input validation or make sure that the `swapTarget` can handle that calldata or that it doesn't revert.

Discussion

Attens1423

SwapTarget and ApproveTarget must be in whitelist maintained by owner. And we expect the routeProxy should not left any tokens. Users should use correct data to avoid gas wasting. We will add ETH check

Issue M-4: Use safeTransferFrom() instead of transferFrom().

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/47

Found by

yixxas, Nyx, 0x4non, Tomo, sach1r0

Summary

The ERC20.transfer() and ERC20.transferFrom() functions return a boolean value indicating success. This parameter needs to be checked for success. Some tokens do not revert if the transfer failed but return false instead.

Vulnerability Detail

Some tokens (like USDT) don't correctly implement the EIP20 standard and their transfer/ transferFrom function return void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will always revert.

Impact

Tokens that don't actually perform the transfer and return false are still counted as a correct transfer and tokens that don't correctly implement the latest EIP20 spec, like USDT, will be unusable in the protocol as they revert the transaction because of the missing return value.

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L420

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L423

Tool used

Manual Review

Recommendation

Recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

Discussion

Evert0x

We think a medium is still valid, although no direct loss of funds, a failed token transfer should be catched.

Issue M-5: Rounding error when call function `dodoMultiswap()` can lead to revert of transaction or fund of user

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/45

Found by

TrungOre

Summary

The calculation of the proportion when do the split swap in function `_multiSwap` doesn't care about the rounding error

Vulnerability Detail

The amount of `midToken` will be transfered to the each adapter can be calculated by formula `curAmount=curTotalAmount*weight/totalWeight`

It will lead to some scenarios when `curTotalAmount*curPoolInfo.weight` is not divisible by `curTotalWeight`, there will be some token left after the swap.

For some tx, if user set a `minReturnAmount` strictly, it may incur the reversion. For some token with small decimal and high value, it can make a big loss for the sender.

Impact

- Revert the transaction because not enough amount of `toToken`
- Sender can lose a small amount of tokens

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L415-L425

Tool used

Manual review

Recommendation

Add a accumulation variable to maintain the total amount is transfered after each split swap. In the last split swap, instead of calculating the `curAmount` by formula above, just take the remaining amount to swap.

Issue M-6: `universalApproveMax` will not work for some tokens that don't support approve `type(uint256).max` amount.

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/41

Found by

jayphbee, Tomo

Summary

`universalApproveMax` will not work for some tokens that don't support approve `type(uint256).max` amount.

Vulnerability Detail

There are tokens that doesn't support approve spender `type(uint256).max` amount. So the `universalApproveMax` will not work for some tokens like `UNI` or `COMP` who will revert when approve `type(uint256).max` amount.

Impact

Tokens that don't support approve `type(uint256).max` amount could not be swapped by calling `externalSwap` function.

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L181-L183

```
if (approveTarget != address(0)) {
    IERC20(fromToken).universalApproveMax(approveTarget, fromTokenAmount);
}
```

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/lib/UniversalERC20.sol#L36-L48

```
function universalApproveMax(
        IERC20 token,
        address to,
        uint256 amount
    ) internal {
        uint256 allowance = token.allowance(address(this), to);
        if (allowance < amount) {
            if (allowance > 0) {
```

```
                token.safeApprove(to, 0);
            }
            token.safeApprove(to, type(uint256).max);
        }
    }
```

Tool used

Manual Review

Recommendation

I would suggest approve only the necessay amount of token to the `approveTarget` instead of the `type(uint256).max` amount.

Issue M-7: `mixSwap` works incorrectly if `mixPairs` length exceed 256.

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/39

Found by

jayphbee

Summary

`mixSwap` works incorrectly if `mixPairs` length exceed 256.

Vulnerability Detail

There's no upper bound check for `mixPairs` length, and the `directions` param whose value will constantly be 0 after 256 rounds right shift. That is to say if the `mixPairs` length exceed 256, the `IDODOAdapter(mixAdapters[i]).sellBase` is always called, which is not the expected behivour.

Impact

`mixSwap` works incorrectly after `mixPairs` length exceed 256.

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L278-L293

```
for (uint256 i = 0; i < mixPairs.length; i++) {
            if (directions & 1 == 0) {
                IDODOAdapter(mixAdapters[i]).sellBase(
                    assetTo[i + 1],
                    mixPairs[i],
                    moreInfos[i]
                );
            } else {
                IDODOAdapter(mixAdapters[i]).sellQuote(
                    assetTo[i + 1],
                    mixPairs[i],
                    moreInfos[i]
                );
            }
            directions = directions >> 1;
        }
```

Tool used

Manual Review

Recommendation

Add upper bound length check for `mixPairs`.

```
require(mixPairs.length <= 256, "DODORouteProxy: PAIRS_LENGTH_TOO_LARGE");
```

Discussion

Attens1423

This is the routeProxy suitable for our own aggregator and users should use right data to avoid gas wasting

Issue M-8: DODORouteProxy#mixSwap doesn't validate all input array lengths

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/31

Found by

ctf_sec, 0x52

Summary

DODORouteProxy#mixSwap validates that most input arrays are the same length but fails to validate the moreInfos array. An array of incorrect length could result in wasted gas on failed swaps or incorrect info being passed to adapters.

Vulnerability Detail

```
require(mixPairs.length > 0, "DODORouteProxy: PAIRS_EMPTY");
require(mixPairs.length == mixAdapters.length, "DODORouteProxy:
↪  PAIR_ADAPTER_NOT_MATCH");
require(mixPairs.length == assetTo.length - 1, "DODORouteProxy:
↪  PAIR_ASSETTO_NOT_MATCH");
require(minReturnAmount > 0, "DODORouteProxy: RETURN_AMOUNT_ZERO");
```

In DODORouteProxy#mixSwap input length checks, moreInfos is missing from the length validation.

Impact

An array of incorrect length could result in wasted gas on failed swaps or incorrect info being passed to adapters leading to unexpected results.

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L238-L311

Tool used

Manual Review

Recommendation

Add a check for moreInfos in the validation block:

```
+   require(mixPairs.length == moreInfos.length, "DODORouteProxy:
↪   PAIR_MOREINFOS_NOT_MATCH");
```

Issue M-9: Hacker can craft malicious 1inch trade to steal the dusted fund in DODORouteProxy.sol

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/26

Found by

ctf_sec

Summary

Hacker can craft malicious 1inch trade to steal the fund in DODORouteProxy.sol

Vulnerability Detail

In DODORouteProxy.sol, We have the superWithdraw function

```
/// @notice used for emergency, generally there wouldn't be tokens left
function superWithdraw(address token) public onlyOwner {
    if(token != _ETH_ADDRESS_) {
        uint256 restAmount = IERC20(token).universalBalanceOf(address(this));
        IERC20(token).universalTransfer(payable(routeFeeReceiver), restAmount);
    } else {
        uint256 restAmount = address(this).balance;
        payable(routeFeeReceiver).transfer(restAmount);
    }
}
```

as the comment suggest, there may be case if the user's trade has dust balance or user send the token to the contract by mistake.

But before the admin can step in a withdraw the fund, a hacker can step, craft malicious 1inch trade to steal the fund in DODORouteProxy.sol

The attack vector is enabled by multiple traits of the DODORouteProxy.sol:

1. the 1inch router is whitelisted.

As suggested in the comment above the DODORouteProxy.sol

> ExternalSwap is for other routers like 0x, 1inch and paraswap

2. Unlimited allowance is given in the code.

```
require(isApproveWhiteListedContract[approveTarget], "DODORouteProxy: Not
↪  Whitelist Appprove Contract");

// transfer in fromToken
```

```
if (fromToken != _ETH_ADDRESS_) {
    // approve if needed
    if (approveTarget != address(0)) {
        IERC20(fromToken).universalApproveMax(approveTarget, fromTokenAmount);
    }

    IDODOApproveProxy(_DODO_APPROVE_PROXY_).claimTokens(
        fromToken,
        msg.sender,
        address(this),
        fromTokenAmount
    );
}
```

3. 1inch can be used to pull an arbitrary amount of funds from the caller and execute arbitrary call

The design of 1inch's AggregationRouterV4 can be used to pull funds from the DODORouteProxy and execute arbitrary external call:

https://polygonscan.com/address/0x1111111254fb6c44bAC0beD2854e76F90643097d#code#L2309

Please see L2309-2321.

```
if (!srcETH) {
    _permit(address(srcToken), desc.permit);
    srcToken.safeTransferFrom(msg.sender, desc.srcReceiver, desc.amount);
}

{
    bytes memory callData = abi.encodePacked(caller.callBytes.selector,
↪   bytes12(0), msg.sender, data);
    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory result) = address(caller).call{value:
↪   msg.value}(callData);
    if (!success) {
        revert(RevertReasonParser.parse(result, "callBytes failed: "));
    }
}
```

4. The low level call data is supplied by user

Impact

All fund in the contract can be taken before admin can superWithdraw.

Code Snippet

https://polygonscan.com/address/0x1111111254fb6c44bAC0beD2854e76F90643097d#code#L2309

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L158-L224

Tool used

Manual Review

Recommendation

Make sure no fund is left after the transaction is finished, add balanceOf(address(this)) check to make sure there is no dust amount in the contract.

Discussion

Attens1423

we expect the routeProxy should not left any tokens. Users should be responsible for their own tokens.

Issue M-10: Issue when handling native ETH trade and WETH trade in DODO RouterProxy#externalSwap

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/20

Found by

ctf_sec

Summary

Lack of logic to wrap the native ETH to WETH in function externalSwap

Vulnerability Detail

The function exeternalSwap can handle external swaps with 0x, 1inch and paraswap or other external resources.

```
function externalSwap(
    address fromToken,
    address toToken,
    address approveTarget,
    address swapTarget,
    uint256 fromTokenAmount,
    uint256 minReturnAmount,
    bytes memory feeData,
    bytes memory callDataConcat,
    uint256 deadLine
) external payable judgeExpired(deadLine) returns (uint256 receiveAmount) {
    require(isWhiteListedContract[swapTarget], "DODORouteProxy: Not Whitelist
↪   Contract");
    require(isApproveWhiteListedContract[approveTarget], "DODORouteProxy: Not
↪   Whitelist Appprove Contract");

    // transfer in fromToken
    if (fromToken != _ETH_ADDRESS_) {
        // approve if needed
        if (approveTarget != address(0)) {
            IERC20(fromToken).universalApproveMax(approveTarget,
↪   fromTokenAmount);
        }

        IDODOApproveProxy(_DODO_APPROVE_PROXY_).claimTokens(
            fromToken,
            msg.sender,
            address(this),
```

```
            fromTokenAmount
        );
    }

    // swap
    uint256 toTokenOriginBalance;
    if(toToken != _ETH_ADDRESS_) {
        toTokenOriginBalance = IERC20(toToken).universalBalanceOf(address(this));
    } else {
        toTokenOriginBalance = IERC20(_WETH_).universalBalanceOf(address(this));
    }
```

note the code above, if the fromToken is set to _ETH_ADDRESS, indicating the user wants to trade with native ETH pair. the function does has payable modifier and user can send ETH along when calling this function.

However, the toTokenOriginBalance is check the only *WETH* balance instead of ETH balance.

```
if(toToken != _ETH_ADDRESS_) {
    toTokenOriginBalance = IERC20(toToken).universalBalanceOf(address(this));
} else {
    toTokenOriginBalance = IERC20(_WETH_).universalBalanceOf(address(this));
}
```

Then we do the swap:

```
(bool success, bytes memory result) = swapTarget.call{
    value: fromToken == _ETH_ADDRESS_ ? fromTokenAmount : 0
}(callDataConcat);
```

If the fromToken is _ETH_ADDRESS, we send the user supplied fromTokenAmount without verifying that the fromTokenAmount.

Finally, we use the before and after balance to get the amount with received.

```
// calculate toToken amount
  if(toToken != _ETH_ADDRESS_) {
      receiveAmount = IERC20(toToken).universalBalanceOf(address(this)) - (
          toTokenOriginBalance
      );
  } else {
      receiveAmount = IERC20(_WETH_).universalBalanceOf(address(this)) - (
          toTokenOriginBalance
      );
  }
```

We are checking the WETH amount instead of ETH amount again.

The issue is that some trades may settle the trade in native ETH, for example

https://developers.paraswap.network/smart-contracts

we can look into the Paraswap contract

https://etherscan.io/address/0xDEF171Fe48CF0115B1d80b88dc8eAB59176FEe57#writeProxyContract

If we click the implementation contract and see the method swapOnUniswapV2Fork

https://etherscan.io/address/0x4ff0dec5f9a763aa1e5c2a962aa6f4edfee4f9ea#code

Code line 927 - 944, which calls the function

```
function swapOnUniswapV2Fork(
    address tokenIn,
    uint256 amountIn,
    uint256 amountOutMin,
    address weth,
    uint256[] calldata pools
)
    external
    payable
{
    _swap(
        tokenIn,
        amountIn,
        amountOutMin,
        weth,
        pools
    );
}
```

which calls:

```
function _swap(
      address tokenIn,
      uint256 amountIn,
      uint256 amountOutMin,
      address weth,
      uint256[] memory pools
  )
      private
      returns (uint256 tokensBought)
  {
      uint256 pairs = pools.length;
```

```
        require(pairs != 0, "At least one pool required");

        bool tokensBoughtEth;

        if (tokenIn == ETH_IDENTIFIER) {
            require(amountIn == msg.value, "Incorrect msg.value");
            IWETH(weth).deposit{value: msg.value}();
            require(IWETH(weth).transfer(address(pools[0]), msg.value));
        } else {
            require(msg.value == 0, "Incorrect msg.value");
            transferTokens(tokenIn, msg.sender, address(pools[0]), amountIn);
            tokensBoughtEth = weth != address(0);
        }

        tokensBought = amountIn;

        for (uint256 i = 0; i < pairs; ++i) {
            uint256 p = pools[i];
            address pool = address(p);
            bool direction = p & DIRECTION_FLAG == 0;

            tokensBought = NewUniswapV2Lib.getAmountOut(
                tokensBought, pool, direction, p >> FEE_OFFSET
            );
            (uint256 amount0Out, uint256 amount1Out) = direction
                ? (uint256(0), tokensBought) : (tokensBought, uint256(0));
            IUniswapV2Pair(pool).swap(
                amount0Out,
                amount1Out,
                i + 1 == pairs
                    ? (tokensBoughtEth ? address(this) : msg.sender)
                    : address(pools[i + 1]),
                ""
            );
        }

        if (tokensBoughtEth) {
            IWETH(weth).withdraw(tokensBought);
            TransferHelper.safeTransferETH(msg.sender, tokensBought);
        }

        require(tokensBought >= amountOutMin, "UniswapV2Router:
↪  INSUFFICIENT_OUTPUT_AMOUNT");
    }
```

as can clearly see, the code first receive ETH, wrap ETH to WETH, then instead end,

unwrap the WETH to ETH and the send the ETH back to complete the trade.

```
if (tokensBoughtEth) {
    IWETH(weth).withdraw(tokensBought);
    TransferHelper.safeTransferETH(msg.sender, tokensBought);
}
```

In DODORouterProxy.sol#ExternalSwap however, we are using WETH balance before and after to check the received amount,

but if we call swapOnUniswapV2Fork on Paraswap router, the balance change for WETH would be 0

because as we see above, the method on paraswap side wrap ETH to WETH but in the end unwrap WETH and send ETH back.

There is also a lack of a method to wrap the ETH to WETH before the trade. making the ETH-related order not tradeable.

Impact

A lot of method that does not use WETH to settle the trade will not be callable.

Code Snippet

https://github.com/sherlock-audit/2022-11-dodo/blob/main/contracts/SmartRoute/DODORouteProxy.sol#L158-L230

Tool used

Manual Review

Recommendation

We recommend the project change from

```
// swap
uint256 toTokenOriginBalance;
if(toToken != _ETH_ADDRESS_) {
    toTokenOriginBalance = IERC20(toToken).universalBalanceOf(address(this));
} else {
    toTokenOriginBalance = IERC20(_WETH_).universalBalanceOf(address(this));
}
```

```
// swap
uint256 toTokenOriginBalance;
if(toToken != _ETH_ADDRESS_) {
```

```
    toTokenOriginBalance = IERC20(toToken).universalBalanceOf(address(this));
} else {
    toTokenOriginBalance =
↪   IERC20(_ETH_ADDRESS).universalBalanceOf(address(this));
}
```

If we want to use WETH to do the balance check, we can help the user wrap the ETH to WETH by calling before do the balance check.

```
IWETH(_WETH_).deposit(receiveAmount);
```

If we want to use WETH as the reference to trade, we also need to approve external contract to spend our WETH.

We can add

```
if(fromToken == _ETH_ADDRESS) {
   IERC20(_WETH_).universalApproveMax(approveTarget, fromTokenAmount);
}
```

We also need to verify the fromTokenAmount for

```
(bool success, bytes memory result) = swapTarget.call{
    value: fromToken == _ETH_ADDRESS_ ? fromTokenAmount : 0
}(callDataConcat);
```

we can add the check:

```
require(msg.value == fromTokenAmount, "invalid ETH amount");
```

Discussion

Attens1423

In our api, we require toToken is WETH when contructing callData. We will add some notes here. Thanks for noticing

Evert0x

Even tough the API is requiring WETH we still think it's a valid issue as the contract has a payable modifier.

Issue M-11: `call()` should be used instead of `transfer()` on an address payable

Source: https://github.com/sherlock-audit/2022-11-dodo-judging/issues/5

Found by

0xNazgul, yixxas, 8olidity, defsec, Nyx, 0x4non, ElKu, rvierdiiev, Bnke0x0, Tomo, virtualfact, pashov, sach1r0, ak1

Summary

Vulnerability Detail

The `transfer()` and `send()` functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

Impact

The use of the deprecated transfer() function for an address will inevitably make the transaction fail when:

- The claimer smart contract does not implement a payable function.
- The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.
- The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300.
- Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Code Snippet

DODORouteProxy.sol#L152 `payable(routeFeeReceiver).transfer(restAmount);` DODORouteProxy.sol#L489 `payable(msg.sender).transfer(receiveAmount);` UniversalERC20.sol#L29 `to.transfer(amount);`

Tool used

Manual Review

Recommendation

Use `call()` instead of `transfer()`, but be sure to respect the CEI pattern and/or add re-entrancy guards, as several hacks already happened in the past due to this recommendation not being fully understood.

More info on; https://swcregistry.io/docs/SWC-134