Smart Contract Audit

| | |
|---|---|
| **Date** | February 2021 |

# DAOFI

This report contains the results of our engagements with DAOfi in order to review the smart contracts daofi - v1-core & daofi - periphery.

## 1 | EXECUTIVE SUMMARY

This report contains the results of our engagements with DAOfi in order to review the smart contracts daofi - v1-core & daofi - periphery.

Nicholas Ward and Sergii Kravchenko conducted the review over 20 days, February 15 through February 26, 2021.

## 2 | SCOPE

The following commit hashes were used to start our review:

| Repository | Commit Hash |
|---|---|
| daofi-v1-core | 0dfe2caf3a2a7a1b16aff26434f78f0b29491c06 |
| daofi-v1-periphery | fbdbd6aabe235aa01cc2002ef73ceb34776dd857 |

After the end of the first week, review moved to the next commit hashes. All of the findings, recommendations, and recommendations in this report are applicable.

| Repository | Commit Hash |
|---|---|
| daofi-v1-core | 328e6dae9709a93852bb4acb098ea09202702dba |
| daofi-v1-periphery | 5ae517c97d5a12522c33e1c87fdf401b489332fc |

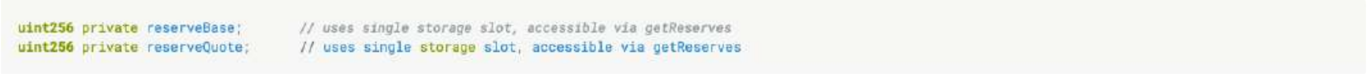
You can find the Appendix with a list of files within scope.

## 3 | RECOMMENDATIONS

### 3.1 | Remove stale comments

Remove inline comments that suggest the two uint256 values DAOfiV1Pair.reserveBase and DAOfiV1Pair.reserveQuote are stored in the same storage slot. This is likely a carryover of the UniswapV2Pair contract in which reserve0 and reserve1 are combined into one storage slot.

**code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L135**

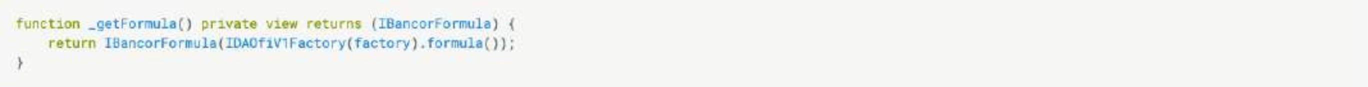


```
uint256 private reserveBase;      // uses single storage slot, accessible via getReserves
uint256 private reserveQuote;     // uses single storage slot, accessible via getReserves
```

### 3.2 Remove unnecessary call to DAOfiV1Factory.formula()

DAOfiV1Pair functions initialize(), getBaseOut() and getQuoteOut() use the private function_getFormula(). This calls the factory to retrieve the BancorFormula contract address. The factory formula address is set in constructor. It cannot be changed. These calls can be replaced by an immutable value in the pair contracts that are set in their constructor.

**code/dao(-v1-core/contracts/DAO(V1Pair.sol:L94-L96**



```
function _getFormula() private view returns (IBancorFormula) {
    return IBancorFormula(IDAOfiV1Factory(factory).formula());
}
```

### 3.3 | Ensure users are aware that the system is incompatible with rebasing and fee-on-transfer tokens

DAOfiV1Pair should never be used with tokens that rebasing. This is tokens in which an account's balance changes with supply expansions and contractions. Funds may be lost because the contract does not provide a mechanism for updating its reserves to respond to unexpected balance adjustments.

DAOfiV1Router01 shouldn't be used with fee on-transfer tokens. This means that tokens in which the recipient of a transfer is not allowed to increase their balance by the amount transferred should not be used. Some router functions have strict controls on the balances, and such tokens would be rejected.

These limitations have been acknowledged by the development team, and it is recommended that users continue to be aware of them.

### 3.4 | Deeper validation of curve math

An increase in testing edge cases in complex mathematical operations may have revealed at least one issue in this report. Additional unit tests, as well property-based or fuzzing testing of curve-related operations are suggested. Incorrectly validated interactions with BancorFormula can lead to unanticipated and potentially deadly failures. Therefore, it is important to validate inputs and avoid pathological curve parameters.

## 4 | FINDINGS

Each issue is assigned a severity

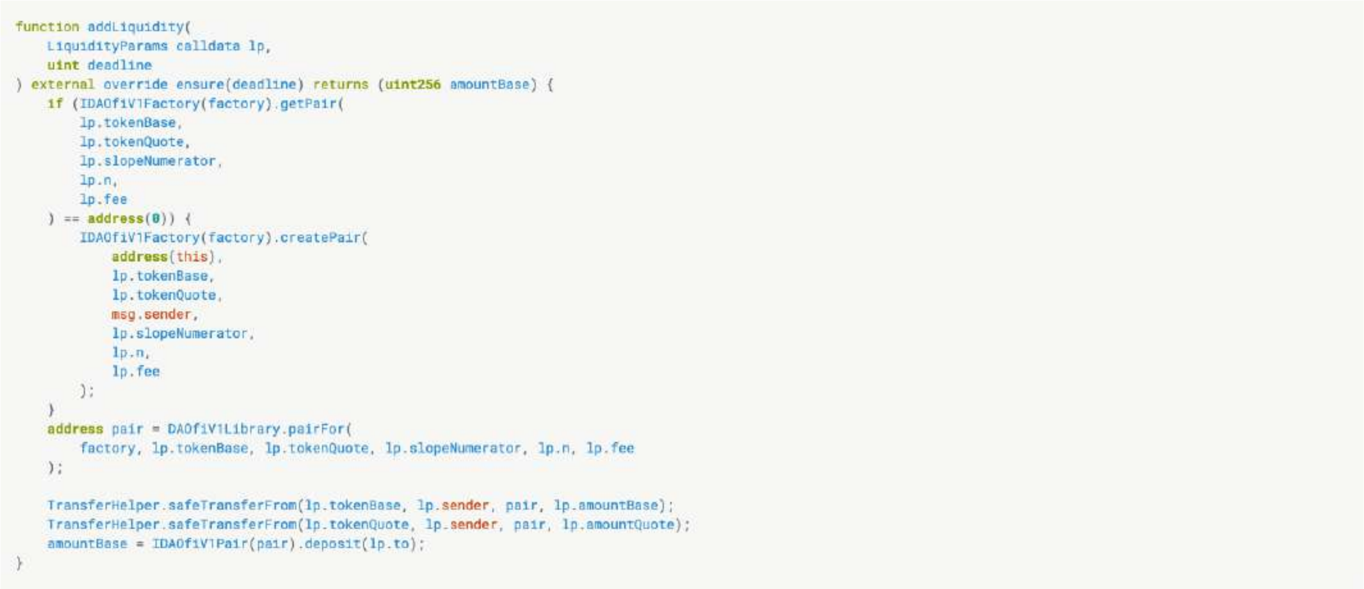
- **Minor** problems are subjective. These are usually suggestions about best practices or readability. These issues should be addressed by code maintainers.
- **Medium** issues are objective, but they are not security vulnerabilities. These issues should be addressed, unless there are compelling reasons not to.
- Security vulnerabilities are critical issues that can't be exploited directly or require special conditions to be exploited. All of these **Major** problems should be addressed.
- Security vulnerabilities that could be exploited to cause **Critical** issues need to be addressed.

### 4.1 | Token approvals can be stolen in DAOfiV1Router01.addLiquidity() **Critical**

Description

DAOfiV1Router01.addLiquidity() creates the desired pair contract if it does not already exist, then transfers tokens into the pair and calls DAOfiV1Pair.deposit(). An attacker could use this method to pass tokens to any address that has received token approvals. This could be used to add liquidity to a pair contract for which the attacker is the pairOwner, allowing the stolen funds to be retrieved using DAOfiV1Pair.withdraw().

**code/dao(-v1-periphery/contracts/DAO(V1Router01.sol:L57-L85**



```
function addLiquidity(
    LiquidityParams calldata lp,
    uint deadline
) external override ensure(deadline) returns (uint256 amountBase) {
    if (IDAOfiV1Factory(factory).getPair(
        lp.tokenBase,
        lp.tokenQuote,
        lp.slopeNumerator,
        lp.n,
        lp.fee
    ) == address(0)) {
        IDAOfiV1Factory(factory).createPair(
            address(this),
            lp.tokenBase,
            lp.tokenQuote,
            msg.sender,
            lp.slopeNumerator,
            lp.n,
            lp.fee
        );
    }
    address pair = DAOfiV1Library.pairFor(
        factory, lp.tokenBase, lp.tokenQuote, lp.slopeNumerator, lp.n, lp.fee
    );

    TransferHelper.safeTransferFrom(lp.tokenBase, lp.sender, pair, lp.amountBase);
    TransferHelper.safeTransferFrom(lp.tokenQuote, lp.sender, pair, lp.amountQuote);
    amountBase = IDAOfiV1Pair(pair).deposit(lp.to);
}
```

### Recommendation

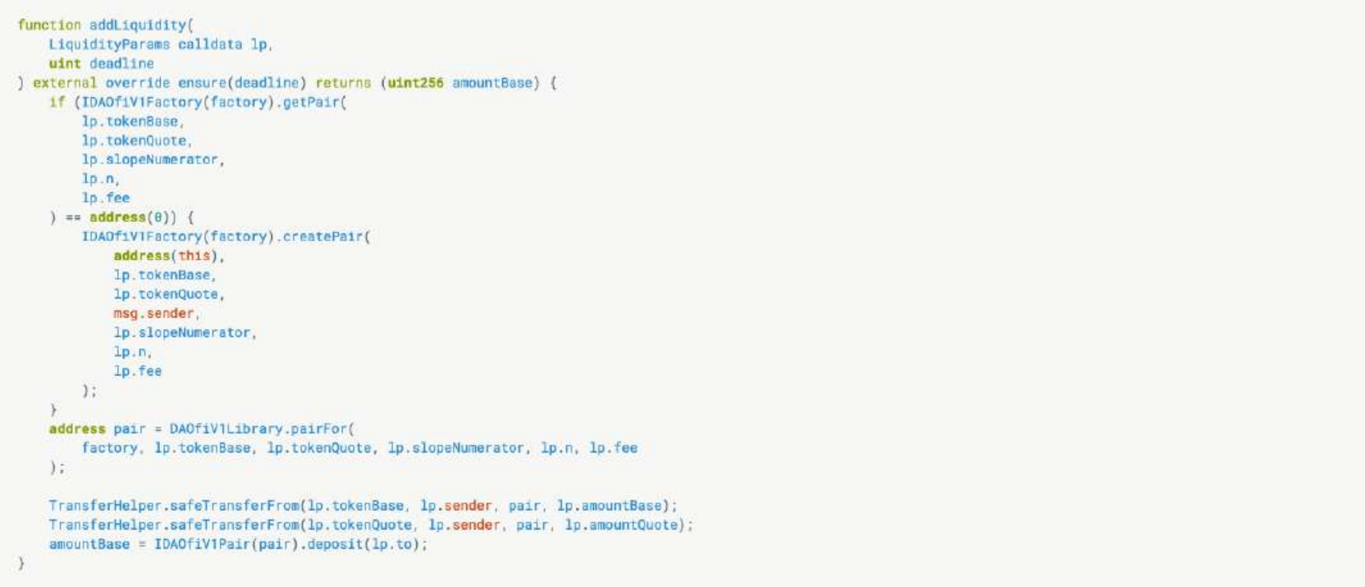
Instead of lp.sender, transfer tokens can be obtained from msg.sender

## 4.2 | The deposit of a new pair can be stolen Critical

### Description

A user must call the same addLiquidity() or addLiquidityETH() function from the router contract to create a new pair.

**code/daofi-v1-periphery/contracts/DAOfiV1Router01.sol:L57-L85**



```
function addLiquidity(
    LiquidityParams calldata lp,
    uint deadline
) external override ensure(deadline) returns (uint256 amountBase) {
    if (IDAOfiV1Factory(factory).getPair(
        lp.tokenBase,
        lp.tokenQuote,
        lp.slopeNumerator,
        lp.n,
        lp.fee
    ) == address(0)) {
        IDAOfiV1Factory(factory).createPair(
            address(this),
            lp.tokenBase,
            lp.tokenQuote,
            msg.sender,
            lp.slopeNumerator,
            lp.n,
            lp.fee
        );
    }
    address pair = DAOfiV1Library.pairFor(
        factory, lp.tokenBase, lp.tokenQuote, lp.slopeNumerator, lp.n, lp.fee
    );

    TransferHelper.safeTransferFrom(lp.tokenBase, lp.sender, pair, lp.amountBase);
    TransferHelper.safeTransferFrom(lp.tokenQuote, lp.sender, pair, lp.amountQuote);
    amountBase = IDAOfiV1Pair(pair).deposit(lp.to);
}
```

This function checks whether the pair exists already and creates one if it doesn't. The first and last deposit are made to the pair.

An attacker can front-run the call and create a new pair with the same parameters (thus with the same address) using the createPair function in the DAOfiV1Factory Contract. The attacker doesn't have to call that function when creating a new pair. This deposit will be made by the initial user, and attacker can withdraw these funds.

### Recommendation

This attack was possible due to a few bugs or factors. All of them or some should be fixed:

| Anyone can call the createPair function of DAOfiV1Factory contract directly without having to deposit with any router address. You could allow only the router create a pair.
| The addLiquidity function verifies that the pair is not yet in existence. If the pair is already owned, the deposit should be made only by the owner.
| However, it is not a good idea to deploy a new pair without making a deposit in the same transaction.

## 4.3 | Incorrect token decimal conversions can lead to loss of funds Major

### Description

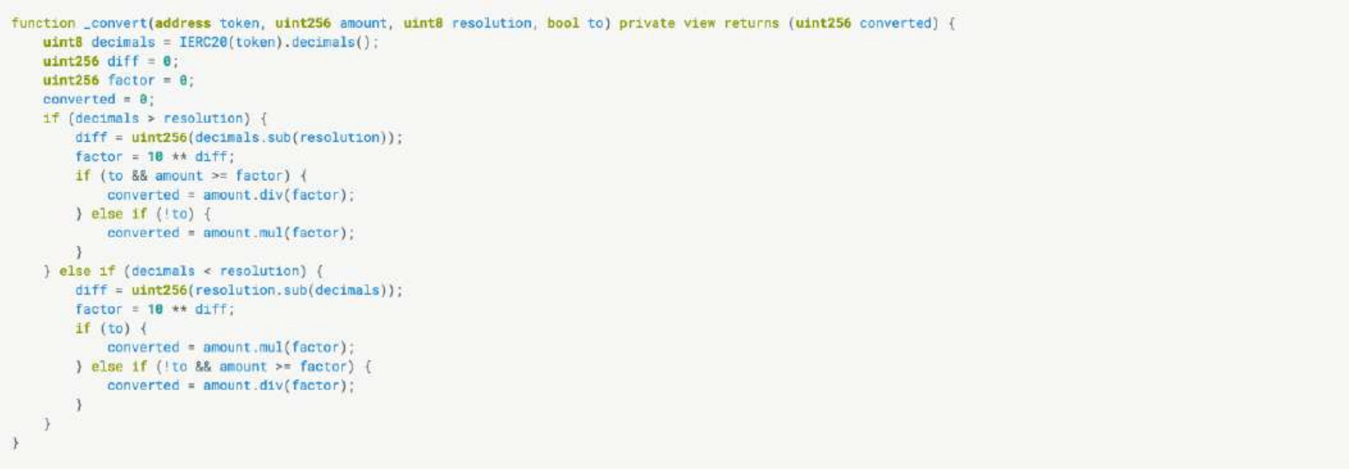
To accommodate tokens that have different decimals() values, the _convert() function of DAOfiV1Pair can be used. It implicitly returns zero for any amount in three situations, the most prominent being when token.decimals() == resolution.

This causes getQuoteOut() to revert whenever baseToken and quoteToken have decimals equal= INTERNAL_DECIMALS (currently set at 8).

GetBaseOut() will also return in most cases where either baseToken nor quoteToken have decimals()= INTERNAL_DECIMALS. GetBaseOut() can only be called when supply is zero, as in deposit(). This causes getBaseOut() not to succeed and returns an incorrect value.

This means that swaps cannot be done in any of these pools. The deposit() function will return an incorrect amountBaseOut baseToken to depositor. The balance can then be withdrawn from the pairOwner.

**code/dao(-v1-core/contracts/DAO(V1Pair.sol:L108-L130**



```
function _convert(address token, uint256 amount, uint8 resolution, bool to) private view returns (uint256 converted) {
    uint8 decimals = IERC20(token).decimals();
    uint256 diff = 0;
    uint256 factor = 0;
    converted = 0;
    if (decimals > resolution) {
        diff = uint256(decimals.sub(resolution));
        factor = 10 ** diff;
        if (to && amount >= factor) {
            converted = amount.div(factor);
        } else if (!to) {
            converted = amount.mul(factor);
        }
    } else if (decimals < resolution) {
        diff = uint256(resolution.sub(decimals));
        factor = 10 ** diff;
        if (to) {
            converted = amount.mul(factor);
        } else if (!to && amount >= factor) {
            converted = amount.div(factor);
        }
    }
}
```

### Recommendation

When token.decimals() is equal to resolution, the _convert() function must return amount. Implicit return values should not be used, especially for functions that perform complex mathematical operations.

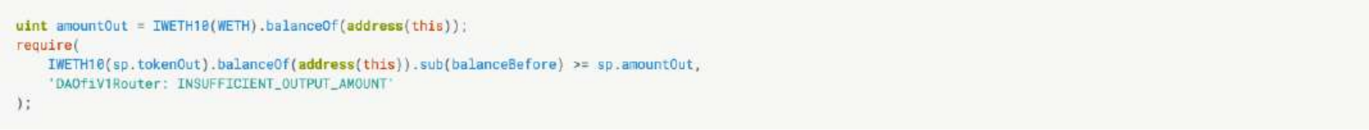
BancorFormula.power(baseN, baseD, _, _) does not support baseN < baseD, and checks should be added to ensure that any call to the BancorFormula conforms to the expected input ranges.

## 4.4 | The cwapExactTokensForETH checks the wrong return value Major

### Description

These lines are meant to verify that the swap tokens received exceed the minimum amount (sp.amountOut).

**code/daofi-v1-periphery/contracts/DAOfiV1Router01.sol:L341-L345**



```
uint amountOut = IWETH10(WETH).balanceOf(address(this));
require(
    IWETH10(sp.tokenOut).balanceOf(address(this)).sub(balanceBefore) >= sp.amountOut,
    'DAOfiV1Router: INSUFFICIENT_OUTPUT_AMOUNT'
);
```

It calculates instead the difference between initial receiver's balance, and balance of router.

### Recommendation

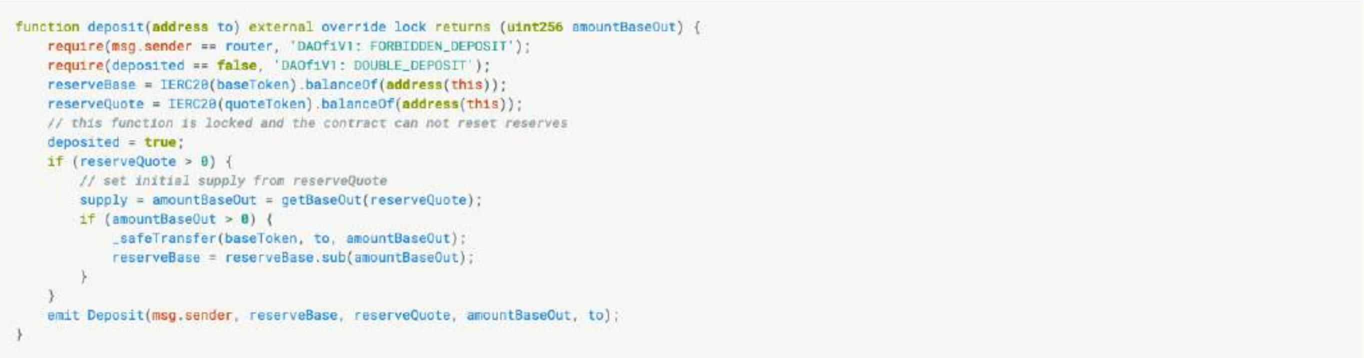
Verify the value you are looking for.

## 4.5 | DAOfiV1Pair.deposit() accepts deposits of zero, blocking the pool Medium

### Description

These lines are meant to verify that the swap tokens received exceed the minimum amount (sp.amountOut).

**code/daofi-v1-periphery/contracts/DAOfiV1Router01.sol:L341-L345**



```
function deposit(address to) external override lock returns (uint256 amountBaseOut) {
    require(msg.sender == router, 'DAOfiV1: FORBIDDEN_DEPOSIT');
    require(deposited == false, 'DAOfiV1: DOUBLE_DEPOSIT');
    reserveBase = IERC20(baseToken).balanceOf(address(this));
    reserveQuote = IERC20(quoteToken).balanceOf(address(this));
    // this function is locked and the contract can not reset reserves
    deposited = true;
    if (reserveQuote > 0) {
        // set initial supply from reserveQuote
        supply = amountBaseOut = getBaseOut(reserveQuote);
        if (amountBaseOut > 0) {
            _safeTransfer(baseToken, to, amountBaseOut);
            reserveBase = reserveBase.sub(amountBaseOut);
        }
    }
    emit Deposit(msg.sender, reserveBase, reserveQuote, amountBaseOut, to);
}
```

### Recommendation

You will need to deposit a minimum amount in baseToken and quoteToken. Do not make assumptions about how baseToken will be distributed as part of your security model.

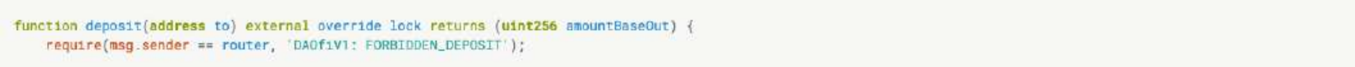
## 4.6 | Restricting DAOfiV1Pair functions to calls from router makes DAOfiV1Router01 security critical Medium

### Description

To avoid any user error, the DAOfiV1Pair functions withdraw(), deposit() and swap() can only be called from the router. This means that any issue in the Router may render all pair contracts useless, possibly locking out the funds of the pair owner.

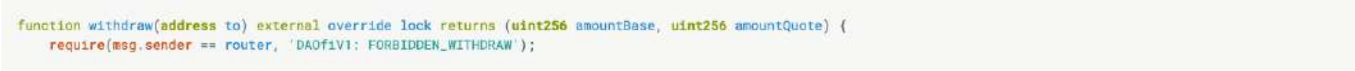
Additionally, DAOfiV1Factory.createPair() allows any nonzero address to be provided as the router, so pairs can be initialized with a malicious router that users would be forced to interact with to utilize the pair contract.

**code/daofi-v1-core/contracts/DAOfiV1Pair.sol:L223-L224**



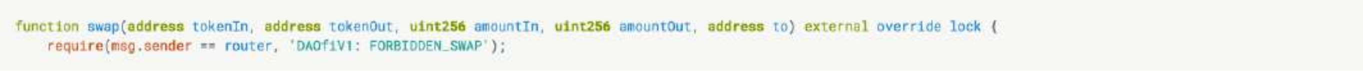
```
function deposit(address to) external override lock returns (uint256 amountBaseOut) {
    require(msg.sender == router, 'DAOfiV1: FORBIDDEN_DEPOSIT');
```

code/dao(-v1-core/contracts/DAO(V1Pair.sol:L250-L251



```
function withdraw(address to) external override lock returns (uint256 amountBase, uint256 amountQuote) {
    require(msg.sender == router, 'DAOfiV1: FORBIDDEN_WITHDRAW');
```

code/dao(-v1-core/contracts/DAO(V1Pair.sol:L292-L293



```
function swap(address tokenIn, address tokenOut, uint256 amountIn, uint256 amountOut, address to) external override lock {
    require(msg.sender == router, 'DAOfiV1: FORBIDDEN_SWAP');
```

### Recommendation

To avoid user error, do not limit DAOfiV1Pair functions only to calls from router. Instead, encourage users to use trusted routers to prevent losses due to user error. This restriction can be kept. You might consider adding the router address to the pair's deployment salt or hardcoding it in DAOfiV1Factory.

## 4.7 | Pair contracts can be easily blocked Minor

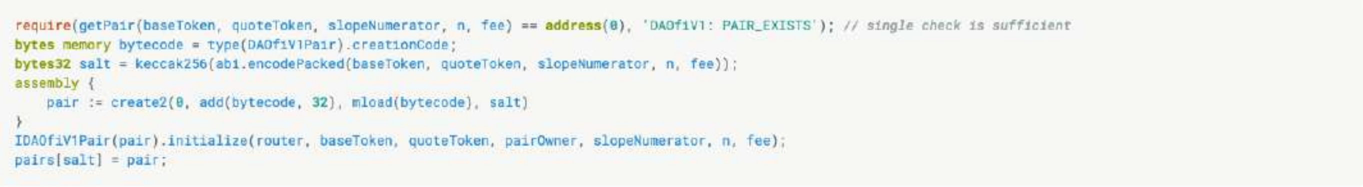
### Description

BaseToken, quoteToken and slopeNumerator are the parameters that define a unique pair. Only one value is accepted for n and eleven for fee. This limits the number of "interesting" pools that can be created for each token pair. Pools can easily be blocked by front-running deployments, depositing zero liquidity immediately or withdrawing any deposited liquidity. These pools cannot be added to again, and are therefore permanently blocked.

This issue can be mitigated by creating a new pool with slightly different parameters. This can lead to significant costs for the pair creator, as they have to deploy a pair that has sub-optimal parameters. It could also block any other pools available for token pairs.

The salt used to determine unique pair contracts in DAOfiV1Factory.createPair():

**code/dao(-v1-core/contracts/DAO(V1Factory.sol:L77-L84**



```
require(getPair(baseToken, quoteToken, slopeNumerator, n, fee) == address(0), 'DAOfiV1: PAIR_EXISTS'); // single check is sufficient
bytes memory bytecode = type(DAOfiV1Pair).creationCode;
bytes32 salt = keccak256(abi.encodePacked(baseToken, quoteToken, slopeNumerator, n, fee));
assembly {
    pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
}
IDAOfiV1Pair(pair).initialize(router, baseToken, quoteToken, pairOwner, slopeNumerator, n, fee);
pairs[salt] = pair;
```

### Recommendation

You might consider adding additional parameters to salt that define a unique pair such as the pairOwner. You can modify the parameters of the salt to partially address any security concerns that were raised in this report.

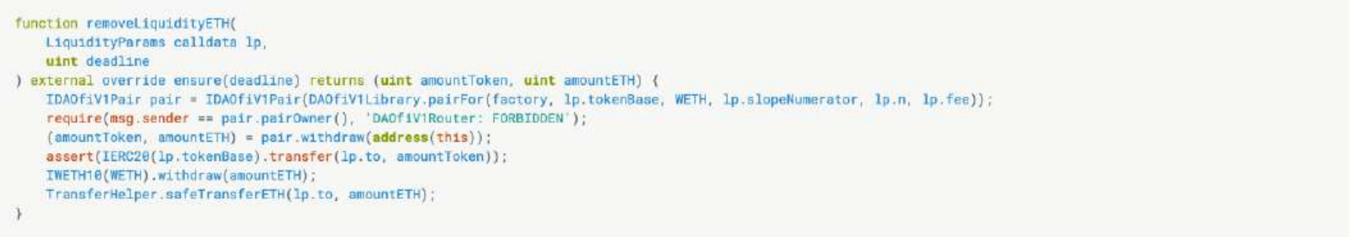
## 4.8 | DAOfiV1Router01.removeLiquidityETH() does not support tokens with no return value Minor

### Description

While the rest of the system uses the safeTransfer* pattern, allowing tokens that do not return a boolean value on transfer() or transferFrom(), DAOfiV1Router01.removeLiquidityETH() throws and consumes all remaining gas if the base token does not return true.

You can withdraw the deposit in this instance without unwrapping it using removeLiquidity().

**code/dao(-v1-periphery/contracts/DAO(V1Router01.sol:L157-L167**



```
function removeLiquidityETH(
    LiquidityParams calldata lp,
    uint deadline
) external override ensure(deadline) returns (uint amountToken, uint amountETH) {
    IDAOfiV1Pair pair = IDAOfiV1Pair(DAOfiV1Library.pairFor(factory, lp.tokenBase, WETH, lp.slopeNumerator, lp.n, lp.fee));
    require(msg.sender == pair.pairOwner(), 'DAOfiV1Router: FORBIDDEN');
    (amountToken, amountETH) = pair.withdraw(address(this));
    assert(IERC20(lp.tokenBase).transfer(lp.to, amountToken));
    IWETH10(WETH).withdraw(amountETH);
    TransferHelper.safeTransferETH(lp.to, amountETH);
}
```

### Recommendation

Be consistent with the use of safeTransfer*, and do not use assert() in cases where the condition can be false.

## APPENDIX 1 - FILES IN SCOPE

The following files were reviewed. SHA-1 hashes were computed for daofi-1-core at commit haveh 328e6da, and daofi-1-periphery at commit hash 5ae517:

| File Name | SHA-1 hash |
|---|---|
| daofi-v1-core/DAOfiV1Pair.sol | a27c969b2716f233dd6c74375c30287628b1dc7b |
| daofi-v1-core/DAOfiV1Factory.sol | 0fef2b496bcd76d9f6824fb7383283edd99e0b60 |
| daofi-v1-core/libraries/SafeMath.sol | 62c7ef91200539f7974c2b6823d77e4c091e59b7 |
| daofi-v1-core/interfaces/IDAOfiV1Pair.sol | 0449a5773b5ba5cc80e8e583c48dbcdf4cac8a91 |
| daofi-v1-core/interfaces/IDAOfiV1Factory.sol | d3727708fb5becfc785b552d792f31dcb824bdea |
| daofi-v1-core/interfaces/IERC20.sol | deeda8921aa5f752effd3ab114d13e9fe46df1e4 |
| daofi-v1-periphery/DAOfiV1Router01.sol | 31c9e9fa1a5c885a83a744d1123292f2ef150de2 |
| daofi-v1-periphery/libraries/DAOfiV1Library.sol | 792df2936dab584bc7e7776052c76e939cf67ad5 |
| daofi-v1-periphery/libraries/SafeMath.sol | 62c7ef91200539f7974c2b6823d77e4c091e59b7 |
| daofi-v1-periphery/interfaces/IERC20.sol | deeda8921aa5f752effd3ab114d13e9fe46df1e4 |
| daofi-v1-periphery/interfaces/IERC2612.sol | 7da8db97d5056bd78c88132dd6a5b3698c965152 |
| daofi-v1-periphery/interfaces/IDAOfiV1Router01.sol | df65a68be60aff44cf666185bb7376d81f776c17 |
| daofi-v1-periphery/interfaces/IWxDAI.sol | 29c8b63b6826e6d297a7692e83637f66a8e3762b |
| daofi-v1-periphery/interfaces/IWETH10.sol | 39ab6ca3cf34d4c90edc468c709eb9aeb52770eb |

## APPENDIX 2 DISCLOSURE

ConsenSys Dialigence ("CD") receives compensation from clients (the Clients) for performing the analysis in these reports (the Reports). Reports can be distributed via ConsenSys publications or other distributions.

Reports are not intended to endorse or indict any project or team. They also do not guarantee security for any project. This Report doesn't consider or have any bearing on the economics of token sales, token sales, or any other product, services, or assets. Cryptographic tokens, which are emerging technologies, carry high technical risks and uncertainties. Any Report does not provide any representation or warranty to Third-Parties in any way. This includes regarding the bug-free nature of code, any business model or proprietors, or the legal compliance of such businesses. The Reports should not be relied upon by any third party, even if it is used to make decisions about buying or selling tokens, products, services, or assets. This Report is not intended as investment advice and should not be relied on as such. It is also not intended to be used as investment advice. CD is not obligated to any Third-Party for publishing these Reports.

PURPOSE OF THE REPORTS Reports and analysis contained therein are only for Clients. They can be published with their permission. Our review will only cover Solidity code. We are limited to reviewing the Solidity codes we have identified as being included in this report. Solidity language is still under development. It may have flaws and risks. The review does NOT cover the compiler layer or any other areas that could pose security risks beyond Solidity. Cryptographic tokens, which are emerging technologies, carry high technical risk and uncertainty.

CD makes the Reports accessible to clients and other parties (i.e. "third parties") via its website. CD hopes that the public availability of these analyses will help the blockchain ecosystem to develop best practices in this rapidly changing area of innovation.

LINKS TO OTHER WEBSITES FROM THIS WEB site You can, via hypertext or other computer hyperlinks, gain access web sites owned by people other than ConsenSys. These hyperlinks are provided only for your convenience and are not intended to replace the owners of these web sites. ConsenSysys or CD are not responsible or liable for any content or operation of these Web sites. You also agree that ConsenSysys or CD will not be liable for the content and/or operation of third-party Web sites. Except as stated below, linking from this Web Site to another site does not mean or imply that ConsenSysys or CD endorses that site's content or its operator. It is up to you to decide whether or not you can use content from any other websites to which the Reports link. ConsenSys or CD will not be responsible for third-party software used on the Web Site. They also assume no liability for any errors or inaccuracies of any output generated by such software.

TIMELINESS CONTENT. The Reports are current as of the Report's date. However, they can be modified at any time. ConsenSys or CD are the only sources of information, unless otherwise indicated.