DATAMI



# QubeSwapRouter, WBNB, QubeSwapFactory, QubeStakeFactory Security Smartcontracts Audit

Prepared by:
The Datami team
account@datami.ua

# Disclaimer

This report includes our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to security vulnerabilities and issues in the smart contract static source code analysed.

In order to get a full view of findings and the scope of analysis, it is crucial to read the full report. While the best efforts were done in conducting the analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against anyone on the basis of what it says or does not say, or how it is produced.

Generally and in addition to this report, it is recommended that the smart contracts undergo additional audits from other teams of auditors and data bug bounty program is conducted on a test network before the deployment of the smart contracts to a production network. Those recommendations are based upon effective security strategies.

This report is not:

- A guarantee of the security of the Binance Smart Chain network.

- A guarantee of future resilience against attacks against the Binance Smart Chain network or other relevant protocol or smart contracts. This includes unknown and undocumented attacks at the time of the audit.

- An absolute determinant of all security issues that may exist within a smart contract.

- A guarantee of the security of the smart contract, if the smart contract is compromised through the owner account, or otherwise designated high permissioned account, maliciously using the contract, whether through a rogue actor or the administrative accounts being compromised.

- Advice regarding the platform through which the smart contract is intended to be deployed.

# Scope

Code of QubeSwapRouter:
*https://bscscan.com/address/0xA8Adb745295e845208E131a3061fB68ed97F55b9#code*

Code of WBNB:
*https://bscscan.com/address/0x356e7fcEbE6CBAA36a5750382753b3e9700cA712#code*

*Code of QubeSwapFactory:*
*https://bscscan.com/address/0x81A0182eE0fd892c0c5DF7e20B4d0b90677657a2#code*

Code of QubeStakeFactory:
*https://bscscan.com/address/0xDa659289C4176b555CD752C49cCf0B46d3F6E41c#code*

The language used is solidity.

QubeSwapRouter:

| Language | files | blank | comment | code |
|---|---|---|---|---|
| Solidity | 1 | 84 | 37 | 712 |

WBNB:

| Language | files | blank | comment | code |
|---|---|---|---|---|
| Solidity | 1 | 136 | 558 | 50 |

QubeSwapFactory:

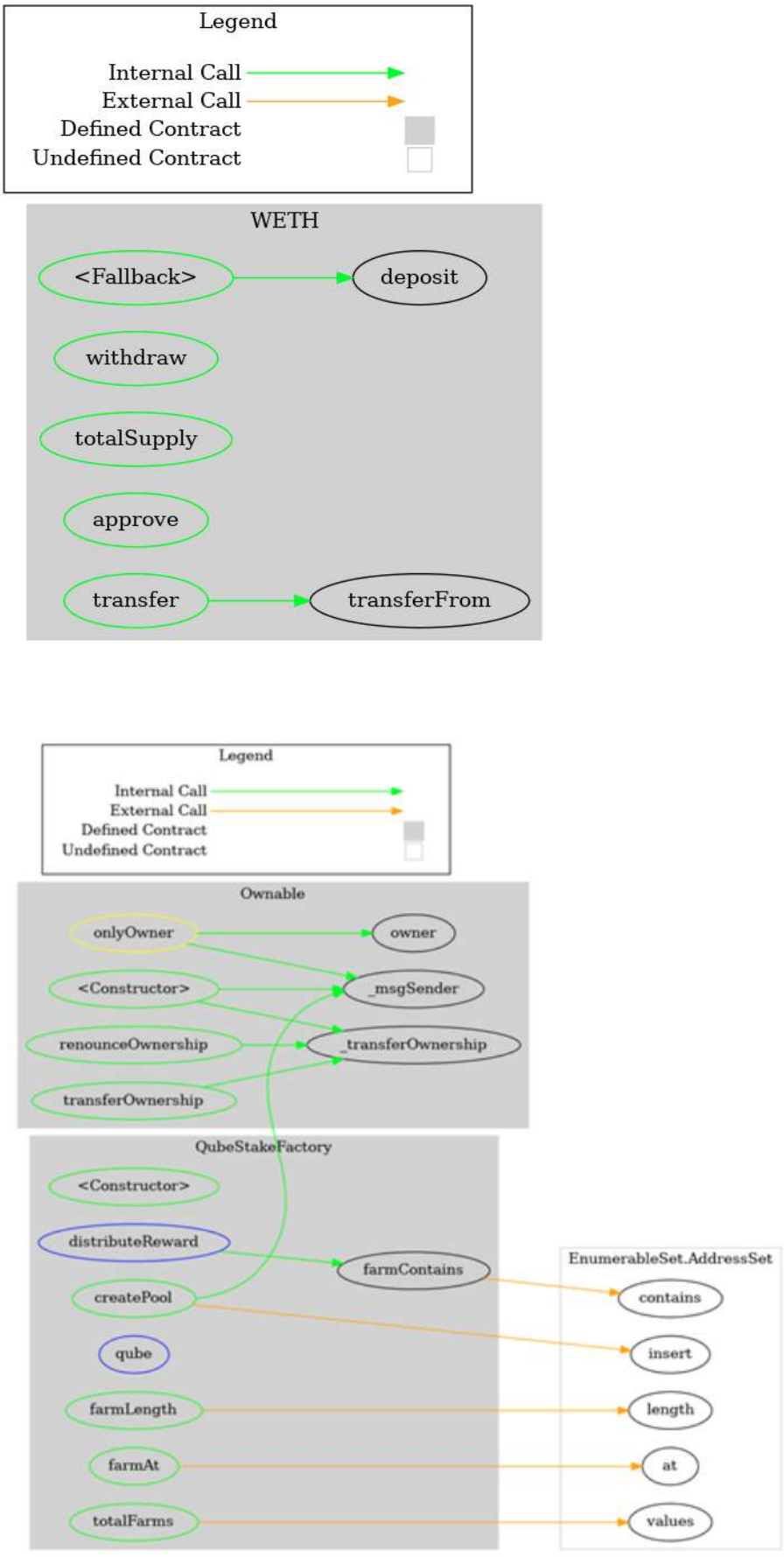
| Language | files | blank | comment | code |
|---|---|---|---|---|
| Solidity | 1 | 80 | 24 | 397 |

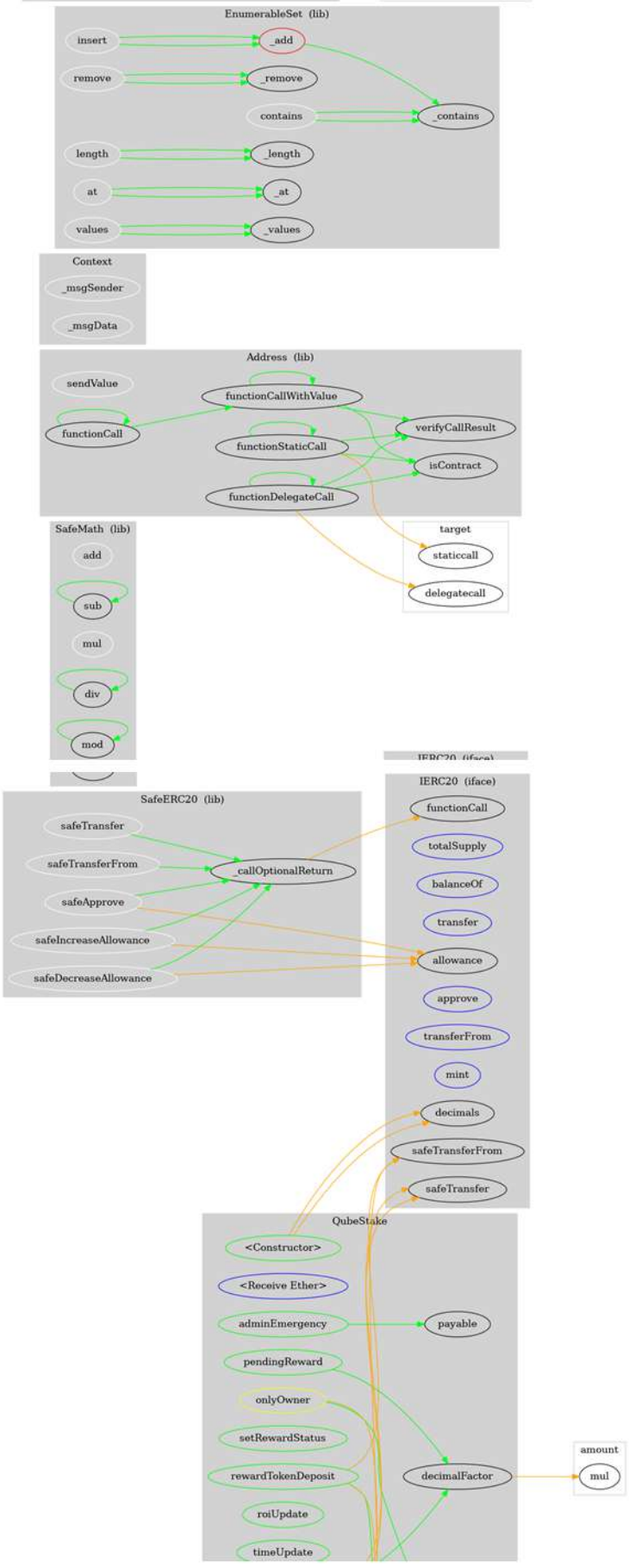
QubeStakeFactory:

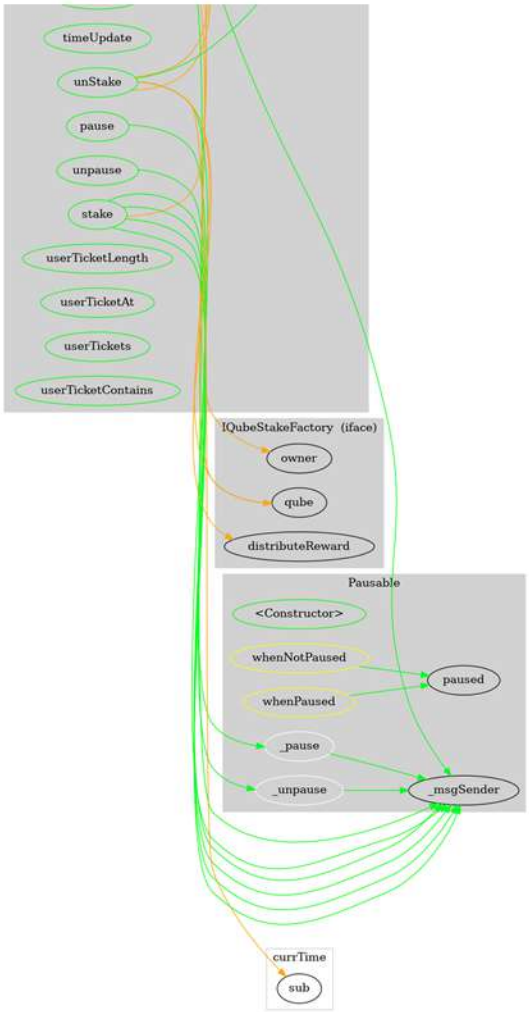
| Language | files | blank | comment | code |
|---|---|---|---|---|
| Solidity | 1 | 162 | 562 | 544 |

DATAMI

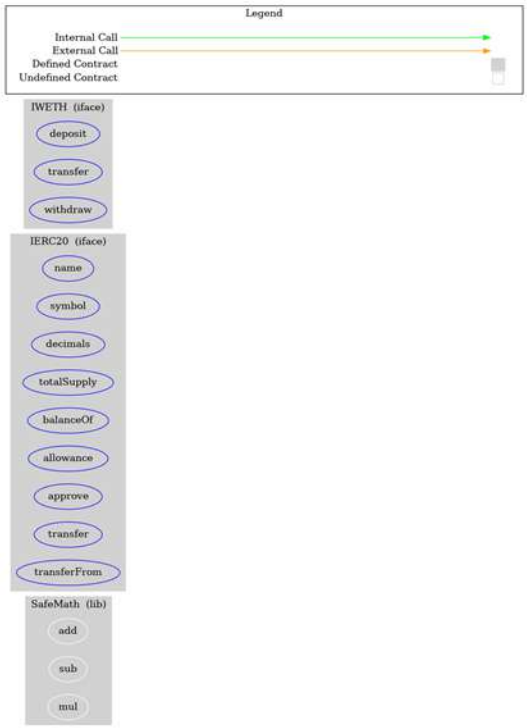
# Diagram:

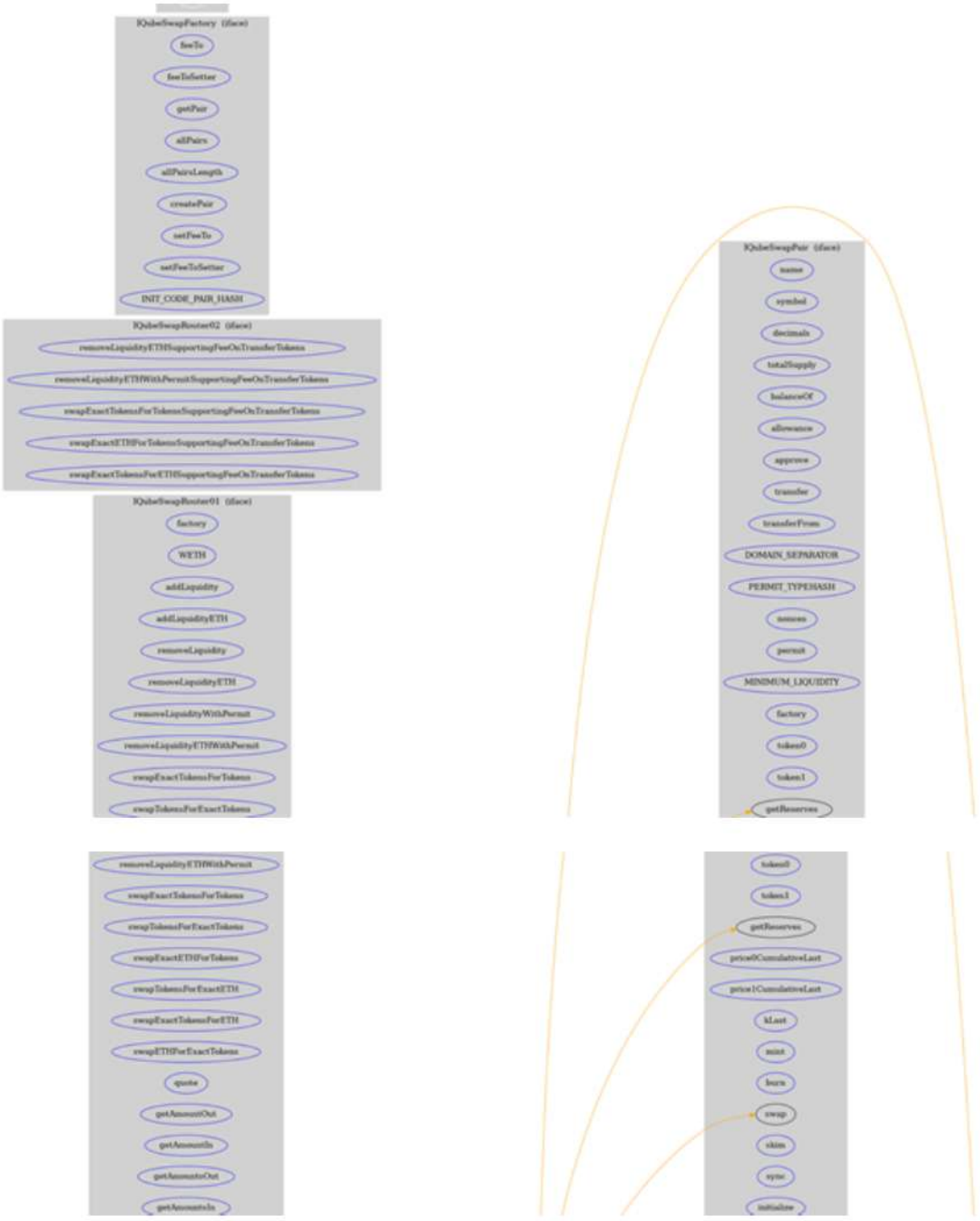

Legend
Internal Call
External Call
Defined Contract
Undefined Contract
WETH
<Fallback>
deposit
withdraw
totalSupply
approve
transfer
transferFrom
Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
QubeStakeFactory
<Constructor>
distributeReward
farmContains
createPool
qube
farmLength
farmAt
totalFarms
EnumerableSet.AddressSet
contains
insert
length
at
values

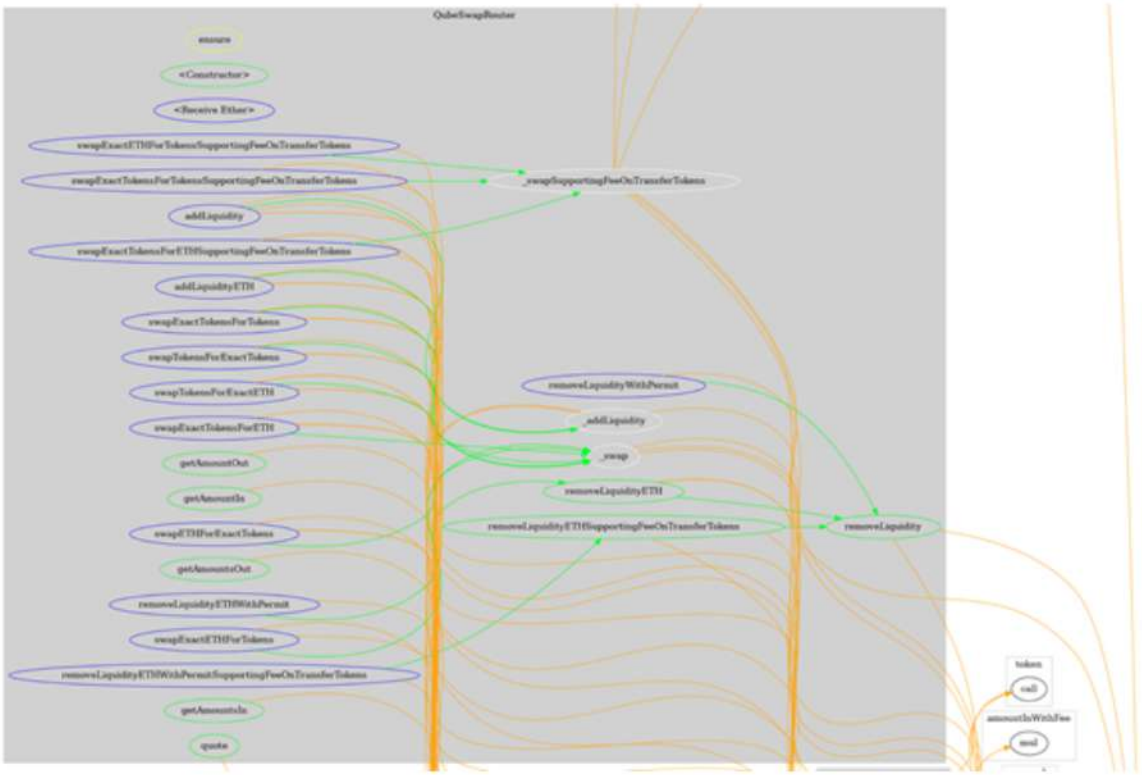

EnumerableSet (lib)
insert
_add
remove
_remove
contains
_contains
length
_length
at
_at
values
_values
Context
_msgSender
_msgData
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
SafeMath (lib)
add
sub
mul
div
mod
IERC20 (iface)
functionCall
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
mint
decimals
safeTransferFrom
safeTransfer
SafeERC20 (lib)
safeTransfer
safeTransferFrom
_callOptionalReturn
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
QubeStake
<Constructor>
<Receive Ether>
adminEmergency
payable
pendingReward
onlyOwner
setRewardStatus
rewardTokenDeposit
decimalFactor
roiUpdate
timeUpdate
amount
mul

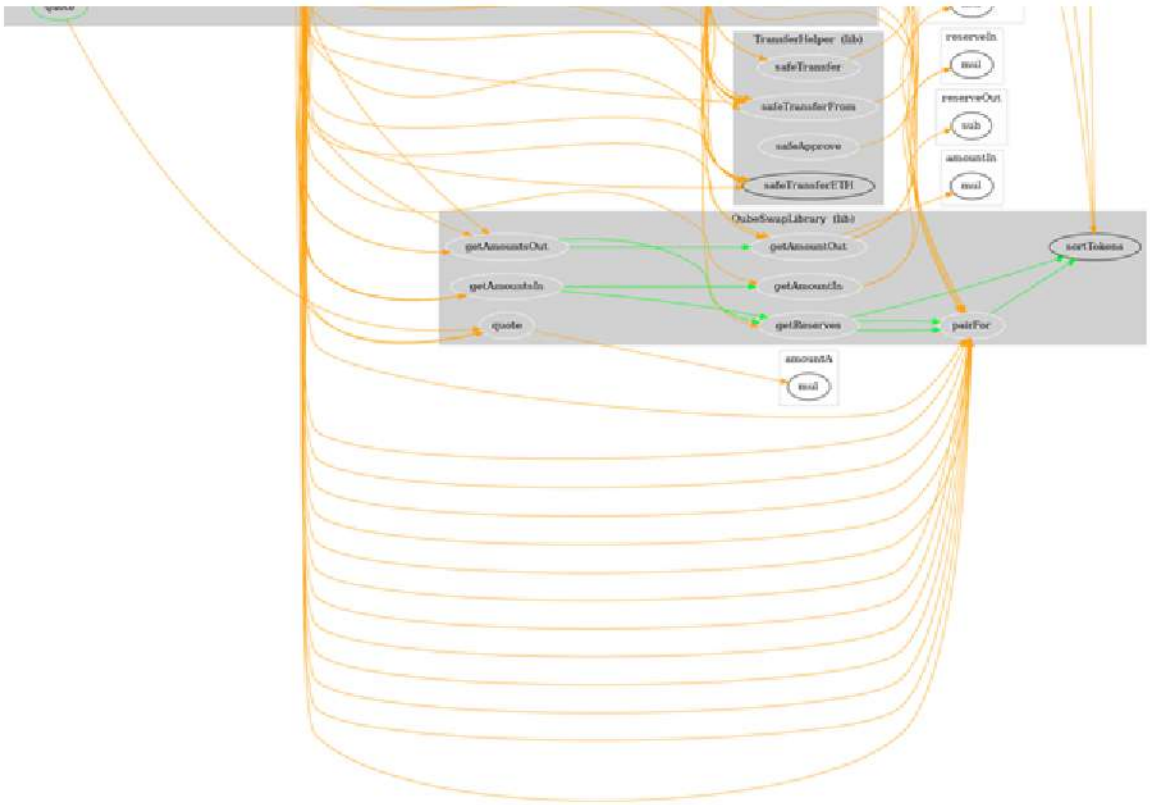



## QubeSwapRouter.sol

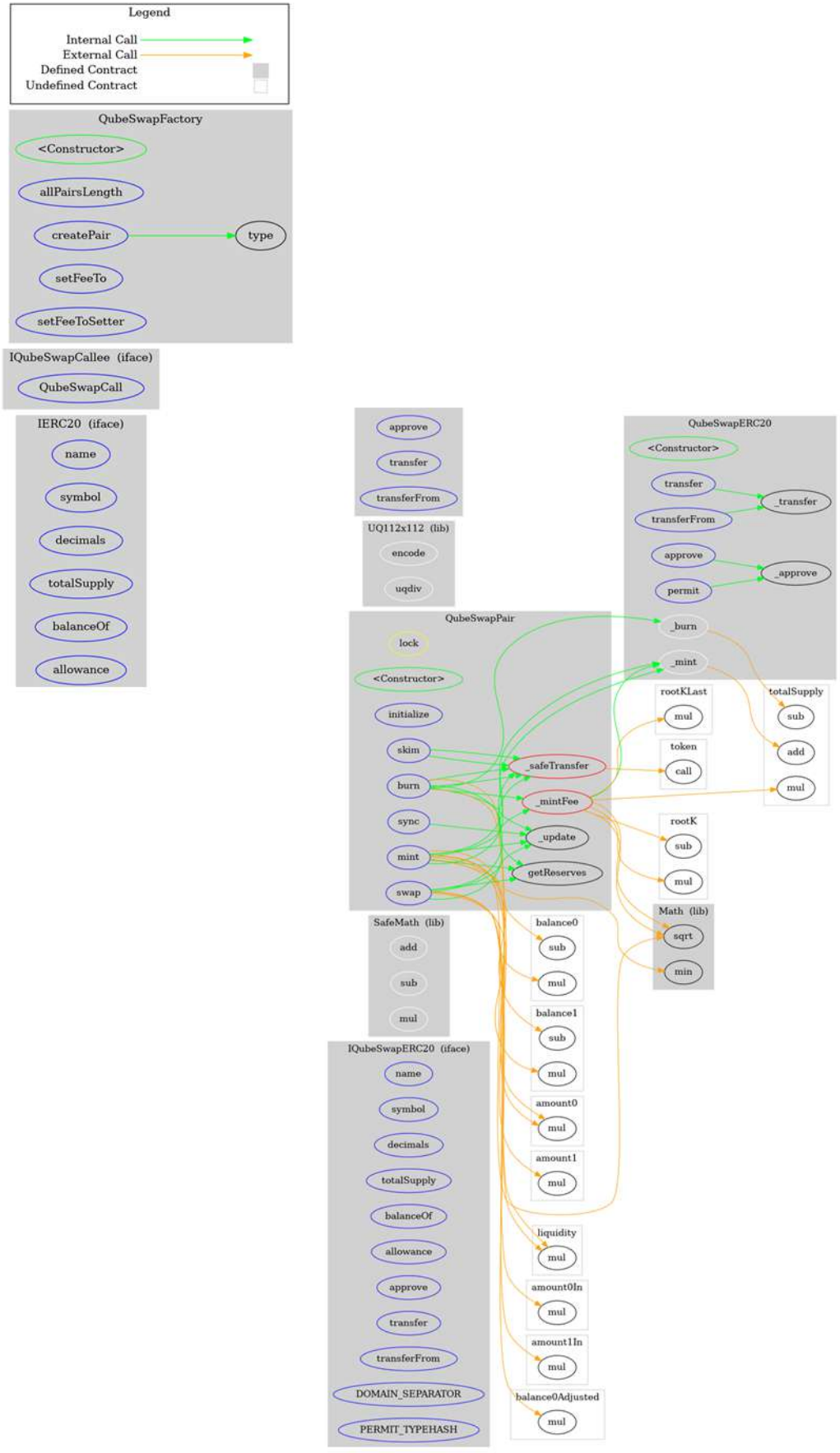


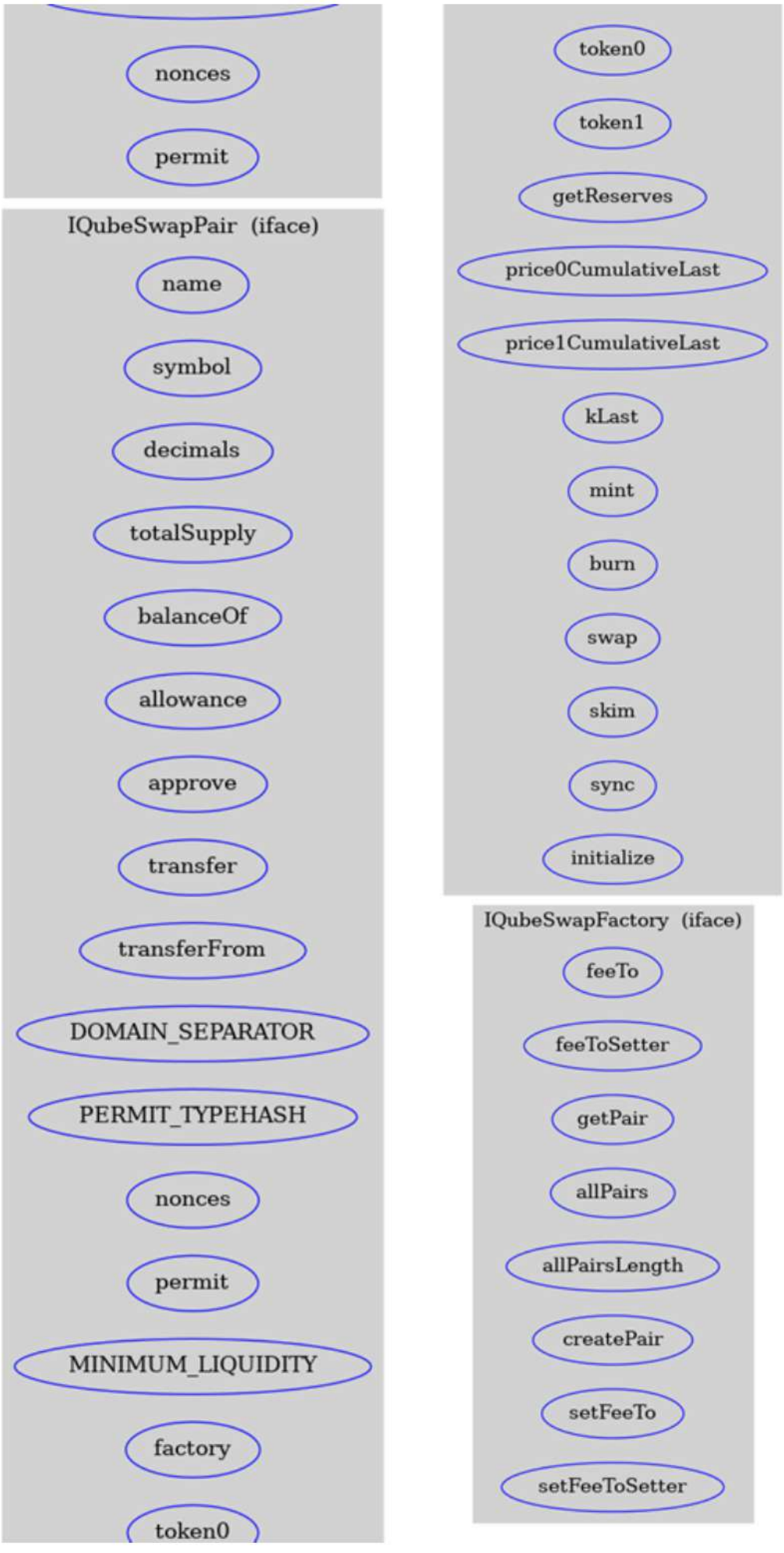

IQubeSwapFactory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
INIT_CODE_PAIR_HASH
IQubeSwapRouter02 (iface)
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
IQubeSwapRouter01 (iface)
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
IQubeSwapPair (iface)
name
symbol
decimals
totalSupply
balanceOf
allowance
approve
transfer
transferFrom
DOMAIN_SEPARATOR
PERMIT_TYPEHASH
nonces
permit
MINIMUM_LIQUIDITY
factory
token0
token1
getReserves
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
token0
token1
getReserves
price0CumulativeLast
price1CumulativeLast
kLast
mint
burn
swap
skim
sync
initialize


QubeSwapRouter
ensure
<Constructor>
<Receive Ether>
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
addLiquidity
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
swapExactTokensForTokens
swapTokensForExactTokens
swapTokensForExactETH
swapExactTokensForETH
getAmountOut
getAmountIn
swapETHForExactTokens
getAmountsOut
removeLiquidityETHWithPermit
swapExactETHForTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
getAmountsIn
quote
_swapSupportingFeeOnTransferTokens
removeLiquidityWithPermit
_addLiquidity
_swap
removeLiquidityETH
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidity
token
call
amountInWithFee
mul


TransferHelper (lib)
safeTransfer
safeTransferFrom
safeApprove
safeTransferETH
reserveIn
mul
reserveOut
sub
amountIn
mul
QubeSwapLibrary (lib)
getAmountsOut
getAmountsIn
quote
getAmountOut
getAmountIn
getReserves
sortTokens
pairFor
amountIA
mul

# QubeSwapFactory.sol




nonces
permit
IQubeSwapPair (iface)
name
symbol
decimals
totalSupply
balanceOf
allowance
approve
transfer
transferFrom
DOMAIN_SEPARATOR
PERMIT_TYPEHASH
nonces
permit
MINIMUM_LIQUIDITY
factory
token0
token0
token1
getReserves
price0CumulativeLast
price1CumulativeLast
kLast
mint
burn
swap
skim
sync
initialize
IQubeSwapFactory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter

# Introduction

The audit was performed between 07.03.2022 - 17.03.2022.

This report is organized into following sections:
1)Executive summary: A high level findings description of the audit.
2)Company Overview.
3)Audit details: A description of the scope and methodology of the audit.
4)Detailed findings of WBNB.
5)Detailed findings of QubeSwapRouter.
6)Detailed findings of QubeSwapFactory.
7)Detailed findings of QubeStakefactory.

The information in this report should be used to understand the risk exposure of the smart contracts and as a guide to improve the security posture of the smart contracts.

# Executive summary

The purpose of the audit was to:
- Conduct security code review of WBNB, QubeSwapRouter, QubeSwapFactory, QubeStakeFactory.
- Identify potential security flaws.

During the assessment, the Cybersecurity Team identified **0** critical vulnerabilities, **0** high risk problems, **0** medium risk vulnerabilities, **16** low risk vulnerabilities and **6** informational aspects.

*Conclusion*
No critical or high vulnerabilities were identified. During the testing process, several low-level and informational vulnerabilities were found, eliminating which can lead to more secure code and following security best practices.

# Company Overview

Datami.ua is a team of 10 cyber security specialists.
We are focused on security testing of applications in various business domains. Datami.ua provides application diagnostics for vulnerabilities, detecting and removal of malicious code, 24/7 protection against possible cyber attacks. In our work we are guided by our own experience, applying the leading standards and approaches of world-class institutes specializing in information security.

# AUDIT DETAILS

A variety of techniques were used to perform the audit.

*Automated Analysis*
Tools were used to automatically detect the presence of potential vulnerabilities, such as reentrancy, timestamp dependency bugs, transaction-ordering dependency bugs, and so on. Static analysis was conducted using Slither.

*Code Review*
Source code was manually reviewed to identify potential security flaws. This type of analysis is useful for detecting business logic flaws and edge-cases but also potential vulnerabilities that may not be detected through static analysis.

Classification of vulnerabilities
Each vulnerability or uncovered risk was ranked on the following steps:
Critical Risk, High Risk, Medium Risk, Low Risk or Informational, they are defined based on the following reasons.

## CRITICAL RISK ISSUES

*These vulnerabilities must be processed instantly due to the high grade of threat they show to the network, users or critical infrastructure.*
*For this kind of vulnerability, using does not require advanced tools or special techniques or advanced knowledge.*

## HIGH RISK ISSUES

*These vulnerabilities must be processed instantly due to the high grade of threat they show for the network, users or data.*
*These vulnerabilities don't require a skilled attacker that possesses advanced tools in order to be exploited, therefore they need to be addressed as soon as possible.*
*Could result in a loss of funds for the contract owner or users.*

## MEDIUM RISK ISSUES

*This vulnerability class needs to be addressed in time.*
*Exploitation is commonly tough and requires social engineering, existing access or special circumstances*
*Results in the code specification operating incorrectly.*

## LOW RISK ISSUES

*These vulnerabilities should be taken into consideration and possessed in the future. These issues offer limited information possibilities to an invader and may not be a real threat.*
*A best practice or design issue that could affect the security standard of the contract.*

## INFORMATIONAL ISSUES

*These are informational disclosure and have very low chances to be used as a real threat.*
*The issue addresses a violation in best practice or a design pattern that has a minimal risk of affecting the security of the contract.*

DATAMI

# Detailed findings of WBNB

## Vulnerability 1 (A floating pragma is set):

*The current pragma Solidity directive is ""^0.4.18"".*

## Risk level:

*Low*

## Recommendations:

It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

```
4
5   pragma solidity ^0.4.18;
6
```

DATAMI

## Vulnerability 2 (Incorrect versions of Solidity):

*Solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statement. Pragma version^0.4.18 (WETH.sol#5) allows old versions solc-0.4.18 is not recommended for deployment.*

## Risk level:

*Info*

## Recommendations:

Deploy with any of the following Solidity versions:

0.5.16 - 0.5.17
0.6.11 - 0.6.12
0.7.5 - 0.7.6
Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

```
4
5   pragma solidity ^0.4.18;
6
```

DATAMI

## Vulnerability 3 (Public function that could be declared external):

*Public functions that are never called by the contract should be declared external to save gas.*

*fallback() should be declared external:*
*- WETH.fallback() (WETH.sol#20-22)*
*withdraw(uint256) should be declared external:*
*- WETH.withdraw(uint256) (WETH.sol#27-32)*
*totalSupply() should be declared external:*
*- WETH.totalSupply() (WETH.sol#34-36)*
*approve(address,uint256) should be declared external:*
*- WETH.approve(address,uint256) (WETH.sol#38-42)*
*transfer(address,uint256) should be declared external:*
*- WETH.transfer(address,uint256) (WETH.sol#44-46)*

## Risk level:

*Info*

## Recommendations:

Use the external attribute for functions never called from the contract.

```
34      function totalSupply() public view returns (uint) {
35          return this.balance;
36      }
```

```
38      function approve(address guy, uint wad) public returns (bool) {
39          allowance[msg.sender][guy] = wad;
40          Approval(msg.sender, guy, wad);
41          return true;
42      }
```

DATAMI

# Detailed findings of QubeSwapRouter

## Vulnerability 1 (Uninitialized local variables):

*Uninitialized local variables.*
*QubeSwapLibrary.getAmountsOut(address,uint256,address[]).i (QubeSwapRouter.sol#341) is a local variable never initialized*
*QubeSwapRouter._swapSupportingFeeOnTransferTokens(address[],address).i (QubeSwapRouter.sol#709) is a local variable never initialized*
*QubeSwapRouter._swap(uint256[],address[],address).i (QubeSwapRouter.sol#600) is a local variable never initialized*

## Risk level:

*Info*

## Recommendations:

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

```
337     function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view returns (uint[] memory amounts) {
338         require(path.length >= 2, 'QubeSwapLibrary: INVALID_PATH');
339         amounts = new uint[](path.length);
340         amounts[0] = amountIn;
341         for (uint i; i < path.length - 1; i++) {
342             (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
343             amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
344         }
345     }
346
```

```
708     function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual {
709         for (uint i; i < path.length - 1; i++) {
710             (address input, address output) = (path[i], path[i + 1]);
711             (address token0,) = QubeSwapLibrary.sortTokens(input, output);
712             IQubeSwapPair pair = IQubeSwapPair(QubeSwapLibrary.pairFor(factory, input, output));
713             uint amountInput;
714             uint amountOutput;
715             { // scope to avoid stack too deep errors
716             (uint reserve0, uint reserve1,) = pair.getReserves();
717             (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
718             amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
719             amountOutput = QubeSwapLibrary.getAmountOut(amountInput, reserveInput, reserveOutput);
720             }
721             (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
722             address to = i < path.length - 2 ? QubeSwapLibrary.pairFor(factory, output, path[i + 2]) : _to;
723             pair.swap(amount0Out, amount1Out, to, new bytes(0));
724         }
```

DATAMI

## Vulnerability 2 (Unused return):

*The return value of an external call is not stored in a local or state variable. QubeSwapRouter._addLiquidity(address,address,uint256,uint256,uint256,uint256) (QubeSwapRouter.sol#420-447) ignores return value by IQubeSwapFactory(factory).createPair(tokenA,tokenB) (QubeSwapRouter.sol#430)*

## Risk level:

*Low*

## Recommendations:

Ensure that all the return values of the function calls are used.

```
420     function _addLiquidity(
421         address tokenA,
422         address tokenB,
423         uint amountADesired,
424         uint amountBDesired,
425         uint amountAMin,
426         uint amountBMin
427     ) internal virtual returns (uint amountA, uint amountB) {
428         // create the pair if it doesn't exist yet
429         if (IQubeSwapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
430             IQubeSwapFactory(factory).createPair(tokenA, tokenB);
431         }
432         (uint reserveA, uint reserveB) = QubeSwapLibrary.getReserves(factory, tokenA, tokenB);
433         if (reserveA == 0 && reserveB == 0) {
434             (amountA, amountB) = (amountADesired, amountBDesired);
435         } else {
436             uint amountBOptimal = QubeSwapLibrary.quote(amountADesired, reserveA, reserveB);
437             if (amountBOptimal <= amountBDesired) {
438                 require(amountBOptimal >= amountBMin, 'QubeSwapRouter: INSUFFICIENT_B_AMOUNT');
439                 (amountA, amountB) = (amountADesired, amountBOptimal);
440             } else {
441                 uint amountAOptimal = QubeSwapLibrary.quote(amountBDesired, reserveB, reserveA);
442                 assert(amountAOptimal <= amountADesired);
443                 require(amountAOptimal >= amountAMin, 'QubeSwapRouter: INSUFFICIENT_A_AMOUNT');
444                 (amountA, amountB) = (amountAOptimal, amountBDesired);
445             }
446         }
447     }
```

## Vulnerability 3 (Missing zero address validation):

*Detect missing zero address validation.*
*QubeSwapRouter.constructor(address,address)._factory (QubeSwapRouter.sol#410) lacks a zero-check on :*
*- factory = _factory (QubeSwapRouter.sol#411)*
*QubeSwapRouter.constructor(address,address)._WETH (QubeSwapRouter.sol#410) lacks a zero-check on :*
*- WETH = _WETH (QubeSwapRouter.sol#412)*

## Risk level:

*Low*

## Recommendations:

Check that the address is not zero.

```
409
410     constructor(address _factory, address _WETH) public {
411         factory = _factory;
412         WETH = _WETH;
413     }
414
```

DATAMI

## Vulnerability 4 (Calls inside a loop):

*Calls inside a loop might lead to a denial-of-service attack.*
*QubeSwapRouter._swap(uint256[],address[],address) (QubeSwapRouter.sol#599-610) has external calls inside a loop: IQubeSwapPair(QubeSwapLibrary.pairFor (factory,input,output)).swap(amount0Out,amount1Out,to,new bytes(0)) (QubeSwapRouter.sol#606-608)*
*QubeSwapRouter._swapSupportingFeeOnTransferTokens(address[],address) (QubeSwapRouter.sol#708-725) has external calls inside a loop: (reserve0,reserve1) = pair.getReserves() (QubeSwapRouter.sol#716)*
*QubeSwapRouter._swapSupportingFeeOnTransferTokens(address[],address) (QubeSwapRouter.sol#708-725) has external calls inside a loop: amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput) (QubeSwapRouter.sol#718)*
*QubeSwapRouter._swapSupportingFeeOnTransferTokens(address[],address) (QubeSwapRouter.sol#708-725) has external calls inside a loop: pair.swap(amount0Out,amount1Out,to,new bytes(0)) (QubeSwapRouter.sol#723)*

## Risk level:

*Low*

## Recommendations:

Favor pull over push strategy for external calls.
https://github.com/ethereum/wiki/wiki/Safety#favor-pull-over-push-for-external-calls

```
599     function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
600         for (uint i; i < path.length - 1; i++) {
601             (address input, address output) = (path[i], path[i + 1]);
602             (address token0,) = QubeSwapLibrary.sortTokens(input, output);
603             uint amountOut = amounts[i + 1];
604             (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut, uint(0));
605             address to = i < path.length - 2 ? QubeSwapLibrary.pairFor(factory, output, path[i + 2]) : _to;
606             IQubeSwapPair(QubeSwapLibrary.pairFor(factory, input, output)).swap(
607                 amount0Out, amount1Out, to, new bytes(0)
608             );
609         }
610     }
```

```
708     function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual {
709         for (uint i; i < path.length - 1; i++) {
710             (address input, address output) = (path[i], path[i + 1]);
711             (address token0,) = QubeSwapLibrary.sortTokens(input, output);
712             IQubeSwapPair pair = IQubeSwapPair(QubeSwapLibrary.pairFor(factory, input, output));
713             uint amountInput;
714             uint amountOutput;
715             { // scope to avoid stack too deep errors
716             (uint reserve0, uint reserve1,) = pair.getReserves();
717             (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
718             amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
719             amountOutput = QubeSwapLibrary.getAmountOut(amountInput, reserveInput, reserveOutput);
720             }
721             (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
722             address to = i < path.length - 2 ? QubeSwapLibrary.pairFor(factory, output, path[i + 2]) : _to;
723             pair.swap(amount0Out, amount1Out, to, new bytes(0));
724         }
725     }
```

```
606                 IQubeSwapPair(QubeSwapLibrary.pairFor(factory, input, output)).swap(
607                     amount0Out, amount1Out, to, new bytes(0)
608                 );
```

DATAMI

## Vulnerability 5 (Low-level calls):

*The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.*
*Low level call in TransferHelper.safeApprove(address,address,uint256) (QubeSwapRouter.sol#11-15):*
*- (success,data) = token.call(abi.encodeWithSelector(0x095ea7b3,to,value)) (QubeSwapRouter.sol#13)*
*Low level call in TransferHelper.safeTransfer(address,address,uint256) (QubeSwapRouter.sol#17-21):*
*- (success,data) = token.call(abi.encodeWithSelector(0xa9059cbb,to,value)) (QubeSwapRouter.sol#19)*
*Low level call in TransferHelper.safeTransferFrom(address,address,address,uint256) (QubeSwapRouter.sol#23-27):*
*- (success,data) = token.call(abi.encodeWithSelector(0x23b872dd,from,to,value)) (QubeSwapRouter.sol#25)*
*Low level call in TransferHelper.safeTransferETH(address,uint256) (QubeSwapRouter.sol#29-32):*
*- (success) = to.call{value: value}(new bytes(0)) (QubeSwapRouter.sol#30)*

## Risk level:

*Info*

## Recommendations:

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

```
11    function safeApprove(address token, address to, uint value) internal {
12        // bytes4(keccak256(bytes('approve(address,uint256)')));
13        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
14        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_FAILED');
15    }
```

```
23    function safeTransferFrom(address token, address from, address to, uint value) internal {
24        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
25        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
26        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_FROM_FAILED');
27    }
```

```
29        function safeTransferETH(address to, uint value) internal {
30            (bool success,) = to.call{value:value}(new bytes(0));
31            require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
32        }
33    }
```

## Vulnerability 6 (Public function that could be declared external):

*Public functions that are never called by the contract should be declared external to save gas.*
*quote(uint256,uint256,uint256) should be declared external:*
*- QubeSwapRouter.quote(uint256,uint256,uint256) (QubeSwapRouter.sol#790-792)*
*getAmountOut(uint256,uint256,uint256) should be declared external:*
*- QubeSwapRouter.getAmountOut(uint256,uint256,uint256) (QubeSwapRouter.sol#794-802)*
*getAmountIn(uint256,uint256,uint256) should be declared external:*
*- QubeSwapRouter.getAmountIn(uint256,uint256,uint256) (QubeSwapRouter.sol#804-812)*
*getAmountsOut(uint256,address[]) should be declared external:*
*- QubeSwapRouter.getAmountsOut(uint256,address[]) (QubeSwapRouter.sol#814-822)*
*getAmountsIn(uint256,address[]) should be declared external:*
*- QubeSwapRouter.getAmountsIn(uint256,address[]) (QubeSwapRouter.sol#824-832)*

## Risk level:

*Info*

## Recommendations:

Use the external attribute for functions never called from the contract.

```
790     function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override returns (uint amountB) {
791         return QubeSwapLibrary.quote(amountA, reserveA, reserveB);
792     }
```

```
794     function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
795         public
796         pure
797         virtual
798         override
799         returns (uint amountOut)
800     {
801         return QubeSwapLibrary.getAmountOut(amountIn, reserveIn, reserveOut);
802     }
```

DATAMI

# Detailed findings of QubeSwapFactory

## Vulnerability 1 (Dangerous strict equalities):

*Use of strict equalities that can be easily manipulated by an attacker.*
*QubeSwapPair._safeTransfer(address,address,uint256) (QubeSwapFactory.sol #297-300) uses a dangerous strict equality:*
*- require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))), QubeSwap: TRANSFER_FAILED) (QubeSwapFactory.sol#299)*
*QubeSwapPair.mint(address) (QubeSwapFactory.sol#363-384) uses a dangerous strict equality:*
*- _totalSupply == 0 (QubeSwapFactory.sol#372)*

## Risk level:

*Low*

## Recommendations:

Don't use strict equality to determine if an account has enough Ether or tokens.

```
297     function _safeTransfer(address token, address to, uint value) private {
298         (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
299         require(success && (data.length == 0 || abi.decode(data, (bool))), 'QubeSwap: TRANSFER_FAILED');
300     }
```

```
363     function mint(address to) external lock returns (uint liquidity) {
364         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
365         uint balance0 = IERC20(token0).balanceOf(address(this));
366         uint balance1 = IERC20(token1).balanceOf(address(this));
367         uint amount0 = balance0.sub(_reserve0);
368         uint amount1 = balance1.sub(_reserve1);
369 
370         bool feeOn = _mintFee(_reserve0, _reserve1);
371         uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
372         if (_totalSupply == 0) {
373             liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
374            _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
375         } else {
376             liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
377         }
378         require(liquidity > 0, 'QubeSwap: INSUFFICIENT_LIQUIDITY_MINTED');
379         _mint(to, liquidity);
380 
381         _update(balance0, balance1, _reserve0, _reserve1);
382         if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
383         emit Mint(msg.sender, amount0, amount1);
384     }
```

## Vulnerability 2 (Reentrancy vulnerabilities):

*Detection of the reentrancy bug. https://github.com/trailofbits/not-so-smart-contracts/tree/master/reentrancy*

*Reentrancy in QubeSwapPair.burn(address) (QubeSwapFactory.sol#387-409):*

**External calls:**

*- _safeTransfer(_token0,to,amount0) (QubeSwapFactory.sol#401)*
*- (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (QubeSwapFactory.sol#298)*
*- _safeTransfer(_token1,to,amount1) (QubeSwapFactory.sol#402)*
*- (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (QubeSwapFactory.sol#298)*

**State variables written after the call(s):**

*- _update(balance0,balance1,_reserve0,_reserve1) (QubeSwapFactory.sol#406)*
*- blockTimestampLast = blockTimestamp (QubeSwapFactory.sol#337)*
*- kLast = uint256(reserve0).mul(reserve1) (QubeSwapFactory.sol#407)*
*- _update(balance0,balance1,_reserve0,_reserve1) (QubeSwapFactory.sol#406)*
*- reserve0 = uint112(balance0) (QubeSwapFactory.sol#335)*
*- _update(balance0,balance1,_reserve0,_reserve1) (QubeSwapFactory.sol#406)*
*- reserve1 = uint112(balance1) (QubeSwapFactory.sol#336)*

*Reentrancy in QubeSwapFactory.createPair(address,address) (QubeSwapFactory.sol#475-490):*

**External calls:**

*- IQubeSwapPair(pair).initialize(token0,token1) (QubeSwapFactory.sol#485)*

*State variables written after the call(s):*

*- getPair[token0][token1] = pair (QubeSwapFactory.sol#486)*
*- getPair[token1][token0] = pair (QubeSwapFactory.sol#487)*

*Reentrancy in QubeSwapPair.swap(uint256,uint256,address,bytes) (QubeSwapFactory.sol#412-440):*

**External calls:**

*- _safeTransfer(_token0,to,amount0Out) (QubeSwapFactory.sol#423)*
*- (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (QubeSwapFactory.sol#298)*
*- _safeTransfer(_token1,to,amount1Out) (QubeSwapFactory.sol#424)*
*- (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (QubeSwapFactory.sol#298)*
*- IQubeSwapCallee(to).QubeSwapCall (msg.sender,amount0Out,amount1Out,data) (QubeSwapFactory.sol#425)*

**State variables written after the call(s):**

*- _update(balance0,balance1,_reserve0,_reserve1) (QubeSwapFactory.sol#438)*
*- blockTimestampLast = blockTimestamp (QubeSwapFactory.sol#337)*
*- _update(balance0,balance1,_reserve0,_reserve1) (QubeSwapFactory.sol#438)*
*- reserve0 = uint112(balance0) (QubeSwapFactory.sol#335)*
*- _update(balance0,balance1,_reserve0,_reserve1) (QubeSwapFactory.sol#438)*
*- reserve1 = uint112(balance1) (QubeSwapFactory.sol#336)*

## Risk level:

*Low*

## Recommendations:

Apply the check-effects-interactions pattern.
http://solidity.readthedocs.io/en/v0.4.21/security-considerations.html#re-entrancy.

```
387     function burn(address to) external lock returns (uint amount0, uint amount1) {
388         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
389         address _token0 = token0;                                // gas savings
390         address _token1 = token1;                                // gas savings
391         uint balance0 = IERC20(_token0).balanceOf(address(this));
392         uint balance1 = IERC20(_token1).balanceOf(address(this));
393         uint liquidity = balanceOf[address(this)];
394 
395         bool feeOn = _mintFee(_reserve0, _reserve1);
396         uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
397         amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
398         amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
399         require(amount0 > 0 && amount1 > 0, 'QubeSwap: INSUFFICIENT_LIQUIDITY_BURNED');
400         _burn(address(this), liquidity);
401         _safeTransfer(_token0, to, amount0);
402         _safeTransfer(_token1, to, amount1);
403         balance0 = IERC20(_token0).balanceOf(address(this));
404         balance1 = IERC20(_token1).balanceOf(address(this));
405 
406         _update(balance0, balance1, _reserve0, _reserve1);
407         if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
408         emit Burn(msg.sender, amount0, amount1, to);
409     }
```

```
475     function createPair(address tokenA, address tokenB) external returns (address pair) {
476         require(tokenA != tokenB, 'QubeSwap: IDENTICAL_ADDRESSES');
477         (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
478         require(token0 != address(0), 'QubeSwap: ZERO_ADDRESS');
479         require(getPair[token0][token1] == address(0), 'QubeSwap: PAIR_EXISTS'); // single check is sufficient
480         bytes memory bytecode = type(QubeSwapPair).creationCode;
481         bytes32 salt = keccak256(abi.encodePacked(token0, token1));
482         assembly {
483             pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
484         }
485         IQubeSwapPair(pair).initialize(token0, token1);
486         getPair[token0][token1] = pair;
487         getPair[token1][token0] = pair; // populate mapping in the reverse direction
488         allPairs.push(pair);
489         emit PairCreated(token0, token1, pair, allPairs.length);
490     }
```

DATAMI

## Vulnerability 3 (Missing zero address validation):

*Detect missing zero address validation.*
*QubeSwapPair.initialize(address,address)._token0 (QubeSwapFactory.sol#319) lacks a zero-check on :*
*- token0 = _token0 (QubeSwapFactory.sol#321)*
*QubeSwapPair.initialize(address,address)._token1 (QubeSwapFactory.sol#319) lacks a zero-check on :*
*- token1 = _token1 (QubeSwapFactory.sol#322)*
*QubeSwapFactory.constructor(address)._feeToSetter (QubeSwapFactory.sol#467) lacks a zero-check on :*
*- feeToSetter = _feeToSetter (QubeSwapFactory.sol#468)*
*QubeSwapFactory.setFeeTo(address)._feeTo (QubeSwapFactory.sol#492) lacks a zero-check on :*
*- feeTo = _feeTo (QubeSwapFactory.sol#494)*
*QubeSwapFactory.setFeeToSetter(address)._feeToSetter (QubeSwapFactory.sol#497) lacks a zero-check on :*
*- feeToSetter = _feeToSetter (QubeSwapFactory.sol#499)*

## Risk level:

*Low*

## Recommendations:

Check that the address is not zero.

```
319     function initialize(address _token0, address _token1) external {
320         require(msg.sender == factory, 'QubeSwap: FORBIDDEN'); // sufficient check
321         token0 = _token0;
322         token1 = _token1;
323     }
```

```
492     function setFeeTo(address _feeTo) external {
493         require(msg.sender == feeToSetter, 'QubeSwap: FORBIDDEN');
494         feeTo = _feeTo;
495     }
496
```

## Vulnerability 4 (Block timestamp):

*Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.*
*QubeSwapERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (QubeSwapFactory.sol#191-203) uses timestamp for comparisons*
*Dangerous comparisons:*
*- require(bool,string)(deadline >= block.timestamp,QubeSwap: EXPIRED) (QubeSwapFactory.sol#192)*
*QubeSwapPair._update(uint256,uint256,uint112,uint112) (QubeSwapFactory.sol#326-339) uses timestamp for comparisons*
*Dangerous comparisons:*
*- timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0 (QubeSwapFactory.sol#330)*

## Risk level:

*Low*

## Recommendations:

Avoid relying on block.timestamp.

```
191     function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external {
192         require(deadline >= block.timestamp, 'QubeSwap: EXPIRED');
193         bytes32 digest = keccak256(
194             abi.encodePacked(
195                 '\x19\x01',
196                 DOMAIN_SEPARATOR,
197                 keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))
198             )
199         );
200         address recoveredAddress = ecrecover(digest, v, r, s);
201         require(recoveredAddress != address(0) && recoveredAddress == owner, 'QubeSwap: INVALID_SIGNATURE');
202         _approve(owner, spender, value);
203     }
204 }
```

```
326     function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
327         require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'QubeSwap: OVERFLOW');
328         uint32 blockTimestamp = uint32(block.timestamp % 2**32);
329         uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
330         if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
331             // * never overflows, and + overflow is desired
332             price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
333             price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
334         }
335         reserve0 = uint112(balance0);
336         reserve1 = uint112(balance1);
337         blockTimestampLast = blockTimestamp;
338         emit Sync(reserve0, reserve1);
339     }
```

## Vulnerability 5 (Assembly usage):

*The use of assembly is error-prone and should be avoided.*
*QubeSwapERC20.constructor() (QubeSwapFactory.sol#134-148) uses assembly*
*- INLINE ASM (QubeSwapFactory.sol#136-138)*
*QubeSwapFactory.createPair(address,address) (QubeSwapFactory.sol#475-490) uses assembly*
*- INLINE ASM (QubeSwapFactory.sol#482-484)*

## Risk level:

*Low*

## Recommendations:

Do not use evm assembly.

```
134     constructor() public {
135         uint chainId;
136         assembly {
137             chainId := chainid
138         }
139         DOMAIN_SEPARATOR = keccak256(
140             abi.encode(
141                 keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)'),
142                 keccak256(bytes(name)),
143                 keccak256(bytes('1')),
144                 chainId,
145                 address(this)
146             )
147         );
148     }
```

```
475     function createPair(address tokenA, address tokenB) external returns (address pair) {
476         require(tokenA != tokenB, 'QubeSwap: IDENTICAL_ADDRESSES');
477         (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
478         require(token0 != address(0), 'QubeSwap: ZERO_ADDRESS');
479         require(getPair[token0][token1] == address(0), 'QubeSwap: PAIR_EXISTS'); // single check is sufficient
480         bytes memory bytecode = type(QubeSwapPair).creationCode;
481         bytes32 salt = keccak256(abi.encodePacked(token0, token1));
482         assembly {
483             pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
484         }
485         IQubeSwapPair(pair).initialize(token0, token1);
486         getPair[token0][token1] = pair;
487         getPair[token1][token0] = pair; // populate mapping in the reverse direction
488         allPairs.push(pair);
489         emit PairCreated(token0, token1, pair, allPairs.length);
490     }
```

DATAMI

## Vulnerability 6 (Low-level calls):

*The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.*
*Low level call in QubeSwapPair._safeTransfer(address,address,uint256) (QubeSwapFactory.sol#297-300):*
*- (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (QubeSwapFactory.sol#298)*

## Risk level:

*Low*

## Recommendations:

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

```
297     function _safeTransfer(address token, address to, uint value) private {
298         (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
299         require(success && (data.length == 0 || abi.decode(data, (bool))), 'QubeSwap: TRANSFER_FAILED');
300     }
```

# Detailed findings of QubeStakeFactory

## Vulnerability 1 (Unused return):

*The return value of an external call is not stored in a local or state variable. QubeStake.stake(uint256) (QubeStakeFactory.sol#1138-1146) ignores return value by userTicketInfo[_msgSender()].insert(internalTicket) (QubeStakeFactory.sol#1141) QubeStakeFactory.createPool(address,address,uint256,uint256,uint256,uint256) (QubeStakeFactory.sol#1220-1241) ignores return value by farms.insert (address(newQube)) (QubeStakeFactory.sol#1240)*

## Risk level:

*Low*

## Recommendations:

Ensure that all the return values of the function calls are used.

```
1138    function stake(uint256 amount) public whenNotPaused {
1139        require(startTime <= block.timestamp && endTime >= block.timestamp, "Stake Expired");
1140        internalTicket++;
1141        userTicketInfo[_msgSender()].insert(internalTicket);
1142        stakeToken.safeTransferFrom(_msgSender(),address(this),amount);
1143        userInfo[internalTicket] = userData(_msgSender(),block.timestamp,block.timestamp.add(yearDuration),block.timestamp,amount,0);
1144
1145        emit stakeEvent(_msgSender(),amount,block.timestamp);
1146    }
```

## Vulnerability 2 (Reentrancy vulnerabilities):

*Detection of the reentrancy bug.*
*https://github.com/trailofbits/not-so-smart-contracts/tree/master/reentrancy*
*Reentrancy in QubeStake.unStake(uint256,uint256) (QubeStakeFactory.sol#1148-1178):*

**External calls:**

*- qubeFactory.distributeReward(userStore.user,getAmountOut) (QubeStakeFactory.sol#1162)*
*- rewardToken.safeTransfer(userStore.user,getAmountOut) (QubeStakeFactory.sol#1164)*

*State variables written after the call(s):*

*- userStore.totalRewards = userStore.totalRewards.add(getAmountOut) (QubeStakeFactory.sol#1166)*
*- userStore.claimTime = currTime (QubeStakeFactory.sol#1169)*

*Reentrancy in QubeStake.unStake(uint256,uint256) (QubeStakeFactory.sol#1148-1178):*

**External calls:**

*- qubeFactory.distributeReward(userStore.user,getAmountOut) (QubeStakeFactory.sol#1162)*
*- rewardToken.safeTransfer(userStore.user,getAmountOut) (QubeStakeFactory.sol#1164)*
*- stakeToken.safeTransfer(userStore.user,amount) (QubeStakeFactory.sol#1173)*

*State variables written after the call(s):*

*- userStore.stakeAmount = userStore.stakeAmount.sub(amount) (QubeStakeFactory.sol#1174)*

## Risk level:

*Low*

## Recommendations:

Apply the check-effects-interactions pattern.
http://solidity.readthedocs.io/en/v0.4.21/security-considerations.html#re-entrancy

```
1148    function unStake(uint256 sid,uint256 amount) public whenNotPaused {
1149        userData storage userStore = userInfo[sid];
1150        require(userTicketInfo[_msgSender()].contains(sid), "Invalid stake id");
1151        require(userStore.stakeAmount >= amount, "invalid amount");
1152
1153        if(rewardState) {
1154            uint256 currTime = userStore.deadLine < block.timestamp ? userStore.deadLine : block.timestamp;
1155            uint256 getAmountOut = (userStore.stakeAmount.mul(
1156                currTime.sub(userStore.claimTime)).mul(
1157                    rewardRoi)).div(100 * yearDuration);
1158
1159            if(getAmountOut > 0){
1160                getAmountOut = decimalFactor(getAmountOut);
1161                if(address(rewardToken) == qubeFactory.qube()){
1162                    qubeFactory.distributeReward(userStore.user,getAmountOut);
1163                }else {
1164                    rewardToken.safeTransfer(userStore.user,getAmountOut);
1165                }
1166                userStore.totalRewards = userStore.totalRewards.add(getAmountOut);
1167                emit rewardEvent(userStore.user,getAmountOut,block.timestamp);
1168            }
1169            userStore.claimTime = currTime;
1170        }
1171
1172        if(amount != 0) {
1173            stakeToken.safeTransfer(userStore.user,amount);
1174            userStore.stakeAmount = userStore.stakeAmount.sub(amount);
1175
1176            emit unstakeEvent(userStore.user,amount,block.timestamp);
1177        }
```

DATAMI

## Vulnerability 3 (Missing zero address validation):

*Detect missing zero address validation.*
*QubeStake.adminEmergency(address,address,uint256).account (QubeStakeFactory.sol#1184) lacks a zero-check on :*
*- address(account).transfer(amount) (QubeStakeFactory.sol#1186)*
*QubeStakeFactory.constructor(address)._qube (QubeStakeFactory.sol#1216) lacks a zero-check on :*
*- qubeToken = _qube (QubeStakeFactory.sol#1217)*

## Risk level:

*Low*

## Recommendations:

Check that the address is not zero.

```
1184    function adminEmergency(address addr,address account,uint256 amount) public onlyOwner {
1185        if(addr == address(0)){
1186            payable(account).transfer(amount);
1187        }else {
1188            IERC20(addr).transfer(account,amount);
1189        }
1190    }
```

```
1216        constructor(address _qube) {
1217            qubeToken = _qube;
1218        }
```

DATAMI

## Vulnerability 4 (Block timestamp):

*Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.*
*QubeStake.stake(uint256) (QubeStakeFactory.sol#1138-1146) uses timestamp for comparisons*
*Dangerous comparisons:*
*- require(bool,string)(startTime <= block.timestamp && endTime >= block.timestamp,Stake Expired) (QubeStakeFactory.sol#1139)*
*QubeStake.unStake(uint256,uint256) (QubeStakeFactory.sol#1148-1178) uses timestamp for comparisons*
*Dangerous comparisons:*
*- require(bool,string)(userStore.stakeAmount >= amount,invalid amount) (QubeStakeFactory.sol#1151)*
*- getAmountOut > 0 (QubeStakeFactory.sol#1159)*
*- userStore.deadLine < block.timestamp (QubeStakeFactory.sol#1154)*

## Risk level:

*Low*

## Recommendations:

Avoid relying on block.timestamp.

```
1138    function stake(uint256 amount) public whenNotPaused {
1139        require(startTime <= block.timestamp && endTime >= block.timestamp, "Stake Expired");
1140        internalTicket++;
1141        userTicketInfo[_msgSender()].insert(internalTicket);
1142        stakeToken.safeTransferFrom(_msgSender(),address(this),amount);
1143        userInfo[internalTicket] = userData(_msgSender(),block.timestamp,block.timestamp.add(yearDuration),block.timestamp,amount,0);
1144
1145        emit stakeEvent(_msgSender(),amount,block.timestamp);
1146    }
```

```
1148    function unStake(uint256 sid,uint256 amount) public whenNotPaused {
1149        userData storage userStore = userInfo[sid];
1150        require(userTicketInfo[_msgSender()].contains(sid), "Invalid Stake id");
1151        require(userStore.stakeAmount >= amount, "invalid amount");
1152
1153        if(rewardState) {
1154            uint256 currTime = userStore.deadLine < block.timestamp ? userStore.deadLine : block.timestamp;
1155            uint256 getAmountOut = (userStore.stakeAmount.mul(
1156                currTime.sub(userStore.claimTime)).mul(
1157                    rewardRoi)).div(100 * yearDuration);
1158
1159            if(getAmountOut > 0){
1160                getAmountOut = decimalFactor(getAmountOut);
1161                if(address(rewardToken) == qubeFactory.qube()){
1162                    qubeFactory.distributeReward(userStore.user,getAmountOut);
1163                }else {
1164                    rewardToken.safeTransfer(userStore.user,getAmountOut);
1165                }
1166                userStore.totalRewards = userStore.totalRewards.add(getAmountOut);
1167                emit rewardEvent(userStore.user,getAmountOut,block.timestamp);
1168            }
1169            userStore.claimTime = currTime;
1170        }
1171
1172        if(amount != 0) {
1173            stakeToken.safeTransfer(userStore.user,amount);
1174            userStore.stakeAmount = userStore.stakeAmount.sub(amount);
1175
1176            emit unstakeEvent(userStore.user,amount,block.timestamp);
1177        }
1178    }
```

## Vulnerability 5 (Assembly usage):

*The use of assembly is error-prone and should be avoided.*

*Address.isContract(address) (QubeStakeFactory.sol#279-289) uses assembly*
*- INLINE ASM (QubeStakeFactory.sol#285-287)*
*Address.verifyCallResult(bool,bytes,string) (QubeStakeFactory.sol#448-468) uses assembly*
*- INLINE ASM (QubeStakeFactory.sol#460-463)*
*EnumerableSet.values(EnumerableSet.AddressSet) (QubeStakeFactory.sol#791-800) uses assembly*
*- INLINE ASM (QubeStakeFactory.sol#795-797)*
*EnumerableSet.values(EnumerableSet.UintSet) (QubeStakeFactory.sol#864-873) uses assembly*
*- INLINE ASM (QubeStakeFactory.sol#868-870)*

## Risk level:

*Low*

## Recommendations:

Do not use evm assembly.

```
279     function isContract(address account) internal view returns (bool) {
280         // This method relies on extcodesize, which returns 0 for contracts in
281         // construction, since the code is only stored at the end of the
282         // constructor execution.
283
284         uint256 size;
285         assembly {
286             size := extcodesize(account)
287         }
288         return size > 0;
289     }
```

```
791     function values(AddressSet storage set) internal view returns (address[] memory) {
792         bytes32[] memory store = _values(set._inner);
793         address[] memory result;
794
795         assembly {
796             result := store
797         }
798
799         return result;
800     }
```

DATAMI

## Vulnerability 6 (Low-level calls):

*The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.*
*Low level call in Address.sendValue(address,uint256) (QubeStakeFactory.sol#307-312):*
*- (success) = recipient.call{value: amount}() (QubeStakeFactory.sol#310)*
*Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (QubeStakeFactory.sol#375-386):*
*- (success,returndata) = target.call{value: value}(data) (QubeStakeFactory.sol#384)*
*Low level call in Address.functionStaticCall(address,bytes,string) (QubeStakeFactory.sol#404-413):*
*- (success,returndata) = target.staticcall(data) (QubeStakeFactory.sol#411)*
*Low level call in Address.functionDelegateCall(address,bytes,string) (QubeStakeFactory.sol#431-440):*
*- (success,returndata) = target.delegatecall(data) (QubeStakeFactory.sol#438)*

## Risk level:

*Low*

## Recommendations:

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

```
307     function sendValue(address payable recipient, uint256 amount) internal {
308         require(address(this).balance >= amount, "Address: insufficient balance");
309 
310         (bool success, ) = recipient.call{value: amount}("");
311         require(success, "Address: unable to send value, recipient may have reverted");
312     }
```

## Vulnerability 7 (Public function that could be declared external):

*Public functions that are never called by the contract should be declared external to save gas.*
*renounceOwnership() should be declared external:*
*- Ownable.renounceOwnership() (QubeStakeFactory.sol#927-929)*
*transferOwnership(address) should be declared external:*
*- Ownable.transferOwnership(address) (QubeStakeFactory.sol#935-938)*
*pause() should be declared external:*
*- QubeStake.pause() (QubeStakeFactory.sol#1106-1108)*
*unpause() should be declared external:*
*- QubeStake.unpause() (QubeStakeFactory.sol#1110-1112)*
*setRewardStatus(bool) should be declared external:*
*- QubeStake.setRewardStatus(bool) (QubeStakeFactory.sol#1114-1116)*
*rewardTokenDeposit(uint256) should be declared external:*
*- QubeStake.rewardTokenDeposit(uint256) (QubeStakeFactory.sol#1118-1121)*
*roiUpdate(uint256) should be declared external:*
*- QubeStake.roiUpdate(uint256) (QubeStakeFactory.sol#1123-1125)*
*timeUpdate(uint256,uint256) should be declared external:*
*- QubeStake.timeUpdate(uint256,uint256) (QubeStakeFactory.sol#1127-1130)*
*pendingReward(uint256) should be declared external:*
*- QubeStake.pendingReward(uint256) (QubeStakeFactory.sol#1132-1136)*
*stake(uint256) should be declared external:*
*- QubeStake.stake(uint256) (QubeStakeFactory.sol#1138-1146)*
*unStake(uint256,uint256) should be declared external:*
*- QubeStake.unStake(uint256,uint256) (QubeStakeFactory.sol#1148-1178)*
*adminEmergency(address,address,uint256) should be declared external:*
*- QubeStake.adminEmergency(address,address,uint256) (QubeStakeFactory.sol#1184-1190)*
*userTicketLength(address) should be declared external:*
*- QubeStake.userTicketLength(address) (QubeStakeFactory.sol#1192-1194)*
*userTicketAt(address,uint256) should be declared external:*
*- QubeStake.userTicketAt(address,uint256) (QubeStakeFactory.sol#1196-1198)*
*userTickets(address) should be declared external:*
*- QubeStake.userTickets(address) (QubeStakeFactory.sol#1200-1202)*
*userTicketContains(address,uint256) should be declared external:*
*- QubeStake.userTicketContains(address,uint256) (QubeStakeFactory.sol#1204-1206)*
*createPool(address,address,uint256,uint256,uint256,uint256) should be declared external:*
*- QubeStakeFactory.createPool(address,address,uint256,uint256,uint256,uint256) (QubeStakeFactory.sol#1220-1241)*
*farmLength() should be declared external:*
*- QubeStakeFactory.farmLength() (QubeStakeFactory.sol#1253-1255)*
*farmAt(uint256) should be declared external:*
*- QubeStakeFactory.farmAt(uint256) (QubeStakeFactory.sol#1257-1259)*
*totalFarms() should be declared external:*
*- QubeStakeFactory.totalFarms() (QubeStakeFactory.sol#1261-1263)*

## Risk level:

*Info*

## Recommendations:

Use the external attribute for functions never called from the contract.

```
927     function renounceOwnership() public virtual onlyOwner {
928         _transferOwnership(address(0));
929     }
```

```
1200    function userTickets(address user) public view returns (uint256[] memory) {
1201        return userTicketInfo[user].values();
1202    }
```

```
1257    function farmAt(uint256 index) public view returns (address) {
1258        return farms.at(index);
1259    }
```

DATAMI

# OUR CONTACTS

**Email:** account@datami.ua

**Phone:**
+38 (099) 613 33 28

**Viber, Telegram, Whatsapp:**
+38 (096) 943 33 28

*DATAMI team*



DATAMI