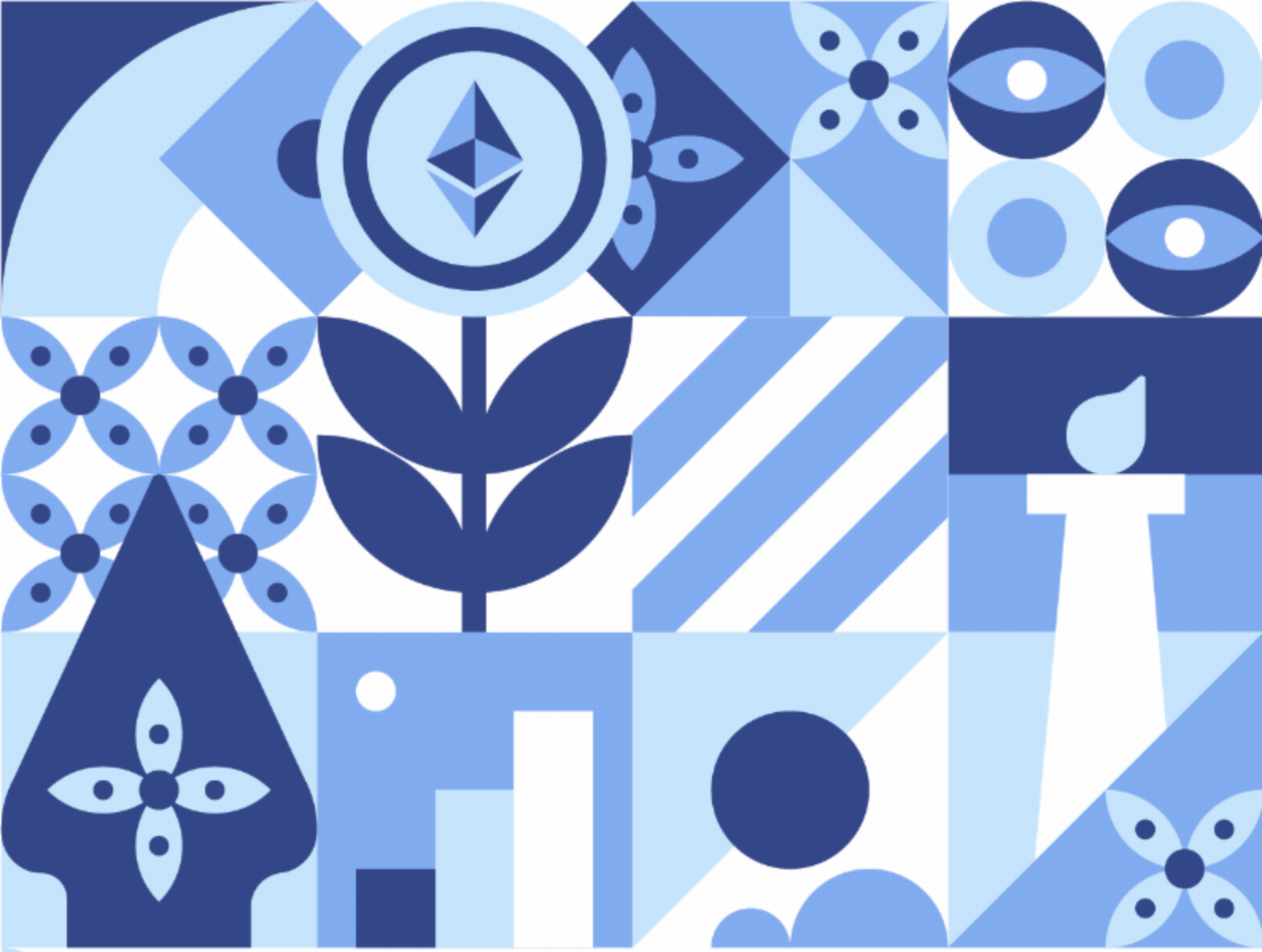

CFX Quantum

Smart Contract Audit Report

SYSFIXED

21 Sep 2020

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
CFX Quantum	CFXQ	Ethereum

Addresses

Contract address	0x0557E0d15aeC0b9026dD17aA874fDf7d182A2cEB
Contract deployer address	0x3c73D73a500373C7689b480a0f7b4b3F35600d52

Project Website

https://cfxquantum.com/

Codebase

https://etherscan.io/address/0x0557E0d15aeC0b9026dD17aA874fDf7d182A2cEB#code

SUMMARY

CFX Quantum is a revolutionary Company that specializes in the most advanced, Trading System, which is ahead it's time.

Contract Summary

Documentation Quality

CFX Quantum provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

Code Quality

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by CFX Quantum with the discovery of several low issues.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-100 SWC-108 | Explicitly define visibility for all state variables on lines 519.
- SWC-102 | It is recommended to use a recent version of the Solidity compiler on lines 5.
- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 78 and 121.
- SWC-111 | It is recommended to use alternatives to the deprecated constructions on lines 14, 15, 25, 250, 311, 365, 445, 455, 467, 568 and 577.
- SWC-115 | tx.origin should not be used for authorization, use msg.sender instead on lines 557, 558, 559, 559, 560, 564 and 564.

CONCLUSION

We have audited the CFX Quantum project released on September 2020 to discover issues and identify potential security vulnerabilities in CFX Quantum Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides a satisfactory result with some low-risk issues.

The issues found in the CFX Quantum smart contract code do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found are an outdated compiler version being used, the use of the "constant" state mutability modifier, an assertion violation was triggered, a state variable visibility is not set as a public state variable with array type causing reachable exception by default, weak sources of randomness, tx.origin as a part of authorization control, and lastly, a requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments)

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	ISSUE FOUND
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	PASS
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	ISSUE FOUND
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	PASS
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	PASS
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	ISSUE FOUND
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	ISSUE FOUND
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	ISSUE FOUND
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	PASS
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Sunday Sep 20 2020 00:48:14 GMT+0000 (Coordinated Universal Time)
Finished	Monday Sep 21 2020 05:29:09 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	Exact Match) Contract Name: CFXQV1.sol

Detected Issues

ID	Title	Severity	Status
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged

SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-115	USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 5

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
4
5   pragma solidity 0.4.26;
6
7   /**
8   * @title ERC20Basic
9
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 519

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "planNumber" is internal. Other possible visibility settings are public and private.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
518
519   uint planNumber = 0;
520
521   mapping(uint => uint) public planTime;
522
523
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 78

low SEVERITY

It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
77   function sub(uint256 a, uint256 b) internal pure returns (uint256) {
78   assert(b <= a);
79   return a - b;
80   }
81
82
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 14

low SEVERITY

Using "constant" as a state mutability modifier in function "totalSupply" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
13   uint public _totalSupply;
14   function totalSupply() public constant returns (uint);
15   function balanceOf(address who) public constant returns (uint);
16   function transfer(address to, uint value) public;
17   event Transfer(address indexed from, address indexed to, uint value);
18
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 15

low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOf" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
14   function totalSupply() public constant returns (uint);
15   function balanceOf(address who) public constant returns (uint);
16   function transfer(address to, uint value) public;
17   event Transfer(address indexed from, address indexed to, uint value);
18   }
19
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 25

low SEVERITY

Using "constant" as a state mutability modifier in function "allowance" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
24   contract ERC20 is ERC20Basic {
25   function allowance(address owner, address spender) public constant returns (uint);
26   function transferFrom(address from, address to, uint value) public;
27   function approve(address spender, uint value) public;
28   event Approval(address indexed owner, address indexed spender, uint value);
29
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 250

low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOf" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
249   */
250   function balanceOf(address _owner) public constant returns (uint balance) {
251   return balances[_owner];
252   }
253
254
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 311

low SEVERITY

Using "constant" as a state mutability modifier in function "allowance" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
310   */
311   function allowance(address _owner, address _spender) public constant returns (uint
remaining) {
312   return allowed[_owner][_spender];
313   }
314
315
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 365

low SEVERITY

Using "constant" as a state mutability modifier in function "getBlackListStatus" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
364   /////// Getters to allow the same blacklist to be used also by other contracts
(including upgraded Tether) ///////
365   function getBlackListStatus(address _maker) external constant returns (bool) {
366   return isBlackListed[_maker];
367   }
368
369
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 445

low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOf" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
444   // Forward ERC20 methods to upgraded contract if this one is deprecated
445   function balanceOf(address who) public constant returns (uint){
446   return super.balanceOf(who);
447   }
448
449
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 455

low SEVERITY

Using "constant" as a state mutability modifier in function "allowance" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
454   // Forward ERC20 methods to upgraded contract if this one is deprecated
455   function allowance(address _owner, address _spender) public constant returns (uint
remaining){
456
457   return super.allowance(_owner, _spender);
458   }
459
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 467

low SEVERITY

Using "constant" as a state mutability modifier in function "totalSupply" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
466   // deprecate current contract if favour of a new one
467   function totalSupply() public constant returns (uint) {
468
469   return _totalSupply;
470   }
471
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 568

low SEVERITY

Using "constant" as a state mutability modifier in function "allPlanAmount" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
567
568   function allPlanAmount(address investor) public constant returns (uint balance){
569
570   uint256 amount = 0;
571   for(uint i = 1; i <= planNumber; i++){
572
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 577

low SEVERITY

Using "constant" as a state mutability modifier in function "planAmount" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
576
577   function planAmount(address investor, uint256 _planNumber) public constant returns
(uint balance){
578
579   return plan[investor][_planNumber];
580   }
581
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 557

low SEVERITY

The tx.origin environment variable has been found to influence a control flow decision. Note that using "tx.origin" as a security control might cause a situation where a user inadvertently authorizes a smart contract to perform an action on their behalf. It is recommended to use "msg.sender" instead.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
556   if(_planTime[i] < block.timestamp){
557   if(_plan[tx.origin][i] > 0){
558   allPlanAmount = allPlanAmount.add(_plan[tx.origin][i]);
559   emit PlanReleased(i, _plan[tx.origin][i], tx.origin);
560   delete plan[tx.origin][i];
561
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 558

low SEVERITY

Using "tx.origin" as a security control can lead to authorization bypass vulnerabilities. Consider using "msg.sender" unless you really know what you are doing.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
557   if(_plan[tx.origin][i] > 0){
558   allPlanAmount = allPlanAmount.add(_plan[tx.origin][i]);
559   emit PlanReleased(i, _plan[tx.origin][i], tx.origin);
560   delete plan[tx.origin][i];
561   }
562
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 559

low SEVERITY

Using "tx.origin" as a security control can lead to authorization bypass vulnerabilities. Consider using "msg.sender" unless you really know what you are doing.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
558   allPlanAmount = allPlanAmount.add(_plan[tx.origin][i]);
559   emit PlanReleased(i, _plan[tx.origin][i], tx.origin);
560   delete plan[tx.origin][i];
561   }
562   }
563
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 559

low SEVERITY

Using "tx.origin" as a security control can lead to authorization bypass vulnerabilities. Consider using "msg.sender" unless you really know what you are doing.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
558   allPlanAmount = allPlanAmount.add(_plan[tx.origin][i]);
559   emit PlanReleased(i, _plan[tx.origin][i], tx.origin);
560   delete plan[tx.origin][i];
561   }
562   }
563
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 560

low SEVERITY

Using "tx.origin" as a security control can lead to authorization bypass vulnerabilities. Consider using "msg.sender" unless you really know what you are doing.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
559   emit PlanReleased(i, _plan[tx.origin][i], tx.origin);
560   delete plan[tx.origin][i];
561   }
562   }
563   }
564
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 564

low SEVERITY

Using "tx.origin" as a security control can lead to authorization bypass vulnerabilities. Consider using "msg.sender" unless you really know what you are doing.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
563   }
564   balances[tx.origin] = balances[tx.origin].add(allPlanAmount);
565
566   }
567
568
```

SWC-115 | USE OF "TX.ORIGIN" AS A PART OF AUTHORIZATION CONTROL.

LINE 564

low SEVERITY

Using "tx.origin" as a security control can lead to authorization bypass vulnerabilities. Consider using "msg.sender" unless you really know what you are doing.

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
563   }
564   balances[tx.origin] = balances[tx.origin].add(allPlanAmount);
565
566   }
567
568
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 121

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- Exact Match) Contract Name: CFXQV1.sol

Locations

```
120   // out and outsize are 0 because we don't know the size yet.
121   let result := delegatecall(gas, implementation, 0, calldatasize, 0, 0)
122   // Copy the returned data.
123   returndatacopy(0, 0, returndatasize)
124   switch result
125
```

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.