QuillAudits

AUDIT REPORT

January 2025

For

okto

Table of Content

Table of Content

Table of Content

Table of Content

Executive Summary

Project Name	Okto milestone 1
Project URL	https://okto.tech/
Overview	Okto milestone 1 contains smart contracts for Okto Orchestration Layer. The set of smart contracts work together with Decentralized Transaction Networks (DTNs) and Decentralized Wallet Networks (DWNs) orchestrate transactions across chains. JobManager.sol is the main contract that provides logic to initiate and terminate the jobs while checking callers/workers are associated with the node. The job can be initiated for any of the supported blocs.for every supported bloc the random node is selected. DTNRegistry.sol contract provides functionality for registering the DTN nodes and adding associated addresses for specific DTN nodes.Additionally it comprises the logic for selecting the random node. BlockRegistry.sol manages the logic for registering the blocs. Bloc contracts like DefiBloc, DerivativeBloc etc can add the supported network and and remove the networks. Only the networks supported by NetworkRegistry are allowed to add. NetworkRegistry is the contract where the protocol admin supported networks which then would be used for validation in other contracts. AccountRegistry allows functionality to store some important data on chain. It stores the account details for vendors and users and provides supporting functionality for it.

Executive Summary

OktoToken is a ERC20 token which is used in protocol for reasons like registering the node in DTNRegistry. It would be also used for rewarding DWNs.

Audit Scope

The scope of this Audit was to analyze the Okto milestone 1 Smart Contracts for quality, security, and correctness.

https://github.com/coindcx-okto/okto-chain-contracts/tree/audits/release-001

Branch : audits/release-001

Contracts in Scope

src\blocs\DeFiBloc.sol
src\blocs\DerivativeBloc.sol
src\blocs\EVMRawTransactionBloc.sol
src\blocs\NFTCreateCollectionBloc.sol
src\blocs\NFTMintBloc.sol
src\blocs\NFTTransferBloc.sol
src\blocs\SwapBloc.sol
src\blocs\TransferBloc.sol

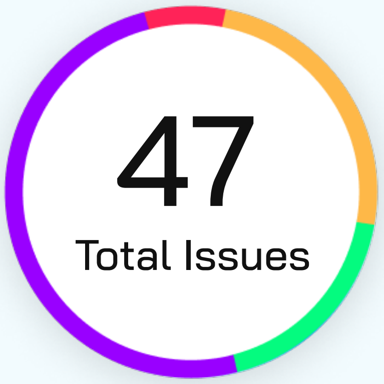
src\contracts\AccountsFactory.sol
src\contracts\BlocRegistry.sol
src\contracts\DTNRegistry.sol
src\contracts\JobManager.sol
src\contracts\NetworkRegistry.sol
src\contracts\OktoTokenTest.sol

src\libraries\NetworkLibrary.sol

Executive Summary

Commit Hash	92d4abe76857f003624cc823f48860becbad9a8b
Language	solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	10th October 2024 - 2nd December 2024
Updated Code Received	17th January 2025 https://github.com/coindcx-okto/okto-chain-contracts/tree/audits/release-final **Comit hash:** 3346baa545c6b8989c0eb6c361805b4f4c78ae64
Review 2	20th January 2025 - 29th January 2025
Fixed In	https://github.com/coindcx-okto/okto-chain-contracts/tree/audit/fix-final **Branch:** fix-final **Comit hash:** a21a2483a32784c841bd075e79f3b17c459fccc8

Number of Issues per Severity



🟥	High	2 (5.0%)
🟧	Medium	13 (27.0%)
🟩	Low	10 (21.2%)
🟪	Informational	22 (46.8%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	1	11	7	19
Acknowledged	1	2	3	3
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls

Checked Vulnerabilities

- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Techniques and Methods

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

🟥 High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

🟧 Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

🟩 Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

🟪 Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	**Resolved** These are the issues identified in the initial audit and have been successfully fixed.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	**Partially Resolved** Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Mapped value of associatedDTN() for specific node worker can be overwritten by anyone

Resolved

Path
DTNRegistry.sol

Function Name
stakeAndRegisterNode()

Description
Mapped value of associatedDTN() for a specific node worker can be overwritten if multiple DTN nodes associate the same worker address.

If this happens then the worker address won't be associated with the previous worker. so functions like isWorkerAssociatedWithNode() return false for the same _nodeAddress and _dtnWorker params for which it was returning true because the address was associated.

Example
1. DTN node1 registers with an array of node workers e.g [worker1,worker2,...]. And the entered node workers get associated with the DTN node1 address.
2. Because the worker (e.g worker2) is associated with DTN node1 address, while terminating job in JobManager the isWorkerAssociatedWithNode() will return true, as it's used to check if the worker is associated with DTN node1 to know that the msg.sender/worker2 is allowed to terminate the job created by DTN node1.
3. Now, DTN node2 also registers itself with the array of node workers e.g [worker80,worker2] which shows that it has the same address which is currently associated with the DTN node1.
4. In the stakeAndRegisterNode() on L93 it will map the associatedDTN[worker2]=node2 address
5. After this if isWorkerAssociatedWithNode() is used by anyone or JobManager to check if the worker2 is associated with the DTN node1 then it will return false because that value is overwritten by node2.

From one perspective It can also be a genuine case where the dtn worker leaves one DTN network and joins another one because of incentives or other things.
This can be also seen as an attack formed by any malicious DTN node (to cause DOS to another DTN node to which a certain worker address was associated).

Recommendation

We recommend verifying and confirming the required logic for this scenario and based on the required logic the code can be modified. We also highlight the need of fixing a case where any DTN node can register and overwrite/steal the worker address associated with the other node.

2. Any worker associated with DTN node can terminate the job because lack of verification

Acknowledged

Path

JobManager.sol

Description

On *L236* the terminateJob() checks that the msg.sender/worker/caller must be associated with the _currentJob.assignedDTN.

But this functionality fails to check if that associated worker can be the address which never executed the transaction/job in actual. Hence any address which is associated with _currentJob.assignedDTN can terminate the job. (this issue also highlights the need of verifying the proofs of transaction while terminating the job)

Recommendation

Proofs can be used to verify if the transaction was actually executed on the target network.

Okto Team's Comments

DTN job execution verification not expected in M1. Any worker can execute the job initiated by any other worker associated to the DTN is expected validation setup can be done in m3.

Medium Severity Issues

3. Updating the GSN params format with updateGsnParamsFormat() can create the DOS

Resolved

Path
JobManager.sol

Function Name
decodeGsnData(), encodeGsnData()

Description
The gsnParamsFormat is used to encode the GSN data offchain so that the encoded data can be passed to initiateJob() where it is decoded using decodeGsnData().

While decoding it with decodeGsnData() the function uses abi.decode() by passing the _data and uses GsnData as type to decode. The *GsnData* is already declared struct on L32.

updateGsnParamsFormat() allows updating the gsnParamsFormat. Let's say after that if initiator initiates the job through initiateJob() then in this case while creating the encoded GSN data offchain is possible by using the updated gsnParamsFormat but when the initiateJob() will use it to decode it with decodeGsnData() the transaction will revert because of different data type which will create denial-of-service (DoS) which won't allow initiating the job with updated gsnParamsFormat.

Additionally encodeGsnData() would not be useful as it takes parameters of type GsnData which won't match the length and type of the new params format.

Recommendation
Due to the identified issue, we recommend against implementing this functionality for updating GSN params format. Additionally we encourage the Okto team to explore alternative solutions.

4. Handle the case when the selected DTN node never executes the job

Acknowledged

Path

JobManager.sol

Description

There can be a case where the DTN node does not execute the job. In that case it can be handled by adding functionality to penalize the DTN node to avoid such scenarios. Additionally we also highlight the possible future need for letting other DTN nodes execute that job in a systematic way in these cases.

Recommendation

Verify the business logic and modify the code according to the requirement.

Okto Team's Comment

no changes , assuming DTNs are trusted. Validation and DTN slashing flows can be added in later milestones

5. Anyone can stake and register the node

Acknowledged

Path

DTNRegistry.sol

Description

If anyone can register the node in DTNRegistry with stakeAndRegisterNode(), there's a possibility of some malicious node registering itself and then that node/address can terminate the job without executing the transaction/job on the target chain. there's a possibility of doing malicious operations if that profit (by malicious activity) will surpass the staked amount in the DTN registry. And If the stake amount is not removable then there's nothing to lose for that malicious DTN node.

Recommendation

Consider rethinking the logic for stakeAndRegisterNode() function about who should be able to stake and register their node. Add the functionality to check only authorized addresses can register.

Okto Team's Comment

currently anyone can come and register, since it requires staking and staking amount determines the reputation score for dtn slection , there are lesser chances of spam .Approval flow will be added in later milestones

6. Precision loss while calculating reputationScore

Resolved

Path
DTNRegistry.sol

Function Name
stakeAndRegisterNode()

Description
stakeAndRegisterNode() calculates _reputationScore based on _amountToStake and minStakeAmount. (_reputationScore = _amountToStake / minStakeAmount)

This calculation will create the precision loss when the result would be in the decimal and because of this the users which entered _amountToStake which caused reputation loss while calculating the _reputationScore will get less reputation score.

Example without precision loss
_amountToStake = 50e18 , minStakeAmount = 1e18
_reputationScore = _amountToStake / minStakeAmount
_reputationScore = 50e18 / 1e18
_reputationScore = 50

Example with precision loss
_amountToStake = 4.99e18 (4990000000000000000) , minStakeAmount = 1e18
_reputationScore = _amountToStake / minStakeAmount
_reputationScore = 4.99e18 / 1e18
_reputationScore = 4.99
The answer would be 4.99 but solidity will only store the 4 and won't store the fractional part.

There can be two ways to avoid precision loss

1. multiply the _amountToStake with 1e18 so the answer would be in decimals and in the wei format as shown below:
   _amountToStake = 4.99e18 (4990000000000000000) , minStakeAmount = 1e18
   _reputationScore = _amountToStake / minStakeAmount
   _reputationScore = (4.99e18 * 1e18) / 1e18
   _reputationScore = 4990000000000000000 (i.e. 4.99e18)

   Note: It should be noted that if the 1e18 is used to multiply the _amountToStake to avoid the precision loss then the answer would be in wei format and it will affect the usage of _reputationScore in other logic. e.g while incrementing the totalReputationScore in stakeAndRegisterNode(), and then usage of totalReputationScore while calculating the randomIndex in _selectRandomNode().

2. Limit the entered _amountToStake value to multiple of minStakeAmount. So in this way there's no possibility of someone entering the values for which the calculated _reputationScore will be in decimal number which will cause the precision loss.
   The example for checking multiple of 1e18 can be:
   if(_amountToStake % minStakeAmount)==0 then its multiple of minStakeAmount.
   In the case it's not multiple of minStakeAmount e.g
   (4.9e18 % 1e18)==0 then it should revert because while calculating the _reputationScore it will cause precision loss.

Recommendation
Fix the precision loss as mentioned in the description.

7. The reduction of factor for failed job is not happening

Resolved

Path
JobManager.sol, DTNRegistry.sol

Description
According to the requirement, the _factor (especially for jobFailedWeight) gets reduced from the allDTNs[_nodeAddress].reputationScore and totalReputationScore. But because of multiple things it's not happening.

1. jobFailedWeight is of uint type so it can't be negative (if the intention was to keep it negative so it will be directly subtracted while adding).
2. And apart from that there's no specific logic which handles this requirement for reducing the factor when the _factor is for jobFailedWeight.

Because of this there's no impact for any DTN node that terminates the job with FAILED status.

Recommendation
Verify the business logic and modify the code to support logic.

8. addNodeWorkers() doesn’t add the new _nodeWorkers to associatedDTN mapping

Resolved

Path
DTNRegistry.sol

Function Name
addNodeWorkers()

Description
addNodeWorkers() doesn’t add the new _nodeWorkers to associatedDTN mapping as it’s happening in stakeAndRegisterNode() on *L93*.
Because of this when isWorkerAssociatedWithNode() is called to check if the DTN worker is associated with a specific DTN address then it will return false.

Recommendation
Consider mapping new _nodeAddress addresses to the _nodeWorkers’s elements as it’s happening in stakeAndRegisterNode().

9. Staked tokens are locked forever

Resolved

Path
DTNRegistry.sol

Description
The node address can stake the amount and register itself and associate the workers with stakeAndRegisterNode(). But there's no way the registered node can remove the tokens.

Recommendation
Consider verifying the logic. Add functionality to remove the tokens.

10. Removing the associated node worker is not possible

Resolved

Path
DTNRegistry.sol

Description
There's no way to remove the worker associated with the DTN node. In some cases it can happen that a specific DTN node worker no longer works for that DTN node but since the JobManager.terminateJob() allows any associated worker to terminate the job.

It creates a vulnerable scenario where a DTN node worker who is not working for a specific DTN can terminate the jobs even while not following that specific DTN's rules.

Recommendation
Verify the business logic and add the logic to unstake the amount if needed.

11. stakeAndRegisterNode() resets the associated values when staked again

Resolved

Path
DTNRegistry.sol

Function Name
stakeAndRegisterNode()

Description
Currently already registered nodes should be able to register again with stakeAndRegisterNode() according to business logic. But while registering again this function resets the values mapped in allDTNs for that msg.sender. Because of this the msg.sender (the DTN node) will lose its important details like stakedAmount, pendingJobs, reputationScore, dtnWorkers.

Additionally in the current stakeAndRegisterNode() implementation the msg.sender address again gets pushed to registeredNodeAddresses array resulting in duplicate addresses.
This process can be repeated multiple times by malicious node addresses to overpopulate the registeredNodeAddresses to increase the chance of getting selected.

Recommendation
In case its expected for the registered node to stake again. instead of assigning the DTN struct again with new values to allDTNs[msg.sender]. values can be added depending on the requirements. e.g _amountToStake can be added to the previous stakedAmount. the pendingJobs can be the same as what it was previously and reputationScore can be added to the previous score and new elements from _nodeWorkers array can be pushed to the previous dtnWorkers array to not lose previously added workers depending on the requirement.

Additionally while pushing elements on L106 it should be checked if the msg.sender is already present in registeredNodeAddresses, it can be checked by if allDTNs[msg.sender] non zero _amountToStake (or any other optimised way).

12. Possible Denial of Service while removing the network supported by bloc

Resolved

Path
NetworkLibrary.sol

Function Name
removeSupportedNetworks()

Description
From the NetworkRegistry, the contract owner can add supported networks allowed in the protocol. Bloc creators consider these supported networks and add them to their bloc in order to work for that network. If in the later future the contract owner removes some network to disallow its support, bloc creators will not be able to remove these networks from their blocs, it reverts if the bloc owner tries to do so. Removing the networks registered by blocs when the network is not available in the networkRegistry which was previously available while adding the network to the bloc can create a DOS.

Example
1. Bloc owner add ("solana") network. while adding the networks the NetworkLibrary validates the network with NetworkRegistry.
2. Let's assume the the "solana" is removed from NetworkRegistry
3. Now block owner wants to remove the "solana" network from the bloc, in this case the NetworkLibrary will validate again that the network is still supported but since the "solana" for example was removed from the NetworkRegistry, L76 in NetworkLibrary.removeSupportedNetworks() will revert.

The impact/severity will vary depending on the intention behind the bloc usage. E.g the user transfers a big amount of funds to another network using that bloc but that network is not supported by networkRegistry (i.e the Okto platform). Depending on how the job would be executed. The funds can get stucked because the platform wasn't supporting that specific network in the first place.

Remediation
Remove the validation while removing the network in the NetworkLibrary.removeSupportedNetworks() on L76 so that currently unsupported network in the NetworkRegistry can be removed from the bloc contract(s).

13. reputationReduction calculation can create a problem

Resolved

Path
DTNRegistry.sol

Function Name
unStake()

Description
If minStakeAmount changes after the node stakes and then the node unstakes, in that case depending on the minStakeAmount is increased or decreased, the reputationReduction will affect the logic.

Example 1: stakedAmount= 10e18 when minStakeAmount was 1e18. Now minStakeAmount is changed to 2e18.
So while unstaking:
reputationReduction = (_amountToUnstake * 1e18) / minStakeAmount
= (10e18 * 1e18) / 2e18
= 5e18
This shows even if the node removed the same/all amount that it staked. The reputationReduction is different i.e. less in this case. The effect of it is the totalReputationScore will be decremented by the less amount even though the whole amount (which was used while incrementing the totalReputationScore) is unstaked.

Example 2: stakedAmount= 10e18 when minStakeAmount was 1e18. Now minStakeAmount is changed to 0.5e18.
So while unstaking:
reputationReduction = (_amountToUnstake * 1e18) / minStakeAmount
= (10e18 * 1e18) / 0.5e18
= 20e18 (the totalReputationScore is 20e18 (or more) at this moment. the additional 10e18 was added when another staker staked.)
In this case, the totalReputationScore will be set to 0 directly as the else block will execute on L250. what it shows is when the minStakeAmount decreases from what it was, the reputationReduction will be more than the calculated reputationScore for that node.
This creates inconsistency in the logic.
Additionally, in Example 2 if the the totalReputationScore goes to 0, The _selectRandomNode() will revert on L193 in else{} because the if{} on L182 will fail.

Recommendation

L249, L250 logic can be replaced to subtract the dtn.reputationScore from totalReputationScore. Additionally, we recommend removing the functionality to adjust minStakeAmount as it introduces multiple issues for maintaining the totalReputationScore.
The edited code should be tested to visit possible paths.

14. The stake amount is not transferred while restaking.

Resolved

Path

DTNRegistry.sol

Function Name

unStake()

Description

The node can register multiple times according to the logic, But when the node stakes next time the if{} will execute but tokens are not transferred because the code to transfer tokens is in else{} i.e. while staking the first time.

This can be used by malicious nodes to unstake more tokens than what was staked.

Example

1. Node staked 1e18
2. The same node stakes again, this time 20e18. Here the nodes stakedAmount would be 21e18 but newly staked 20e18 are not transferred to the contract.
3. Node unstakes. The whole stakedAmount for that node will be unstaked. So node will receive the 21e18.

These 20e18 are transferred from the tokens staked by other users.

In another case, the unstake transaction will revert if there aren’t enough tokens in the contract to transfer to the user.

Recommendation

Transfer the token amount outside of the if{} else{} blocks.

15. Transaction will revert with panic

Resolved

Path
DTNRegistry.sol

Function Name
updateNodeReputationScore()

Description
In updateNodeReputationScore() if the control flow enters if{} on *L219* then the allDTNs[_nodeAddress].reputationScore will be set to 0. Where the _nodeAddress can be assumed as the first element from registeredNodeAddresses in this case.

Going ahead in _updateWeightedScore() on *L387* it subtracts 1 from allDTNs[registeredNodeAddresses[0]].reputationScore which is 0, and the transaction will revert with panic code because of an arithmetic error.

Recommendation
Handle this scenario to preserve the business logic and to avoid the error.

One case to handle this can be to delete the mapped value in allDTNs[] for that node and remove it from registeredNodeAddresses before calling _updateWeightedScore() on L233. But in this case, node will lose its staked amount. We encourage the development team to handle this case to preserve the business logic and to avoid error.

Low Severity Issues

16. Possibility of a node associating node worker which is not a part of it's DTN.

Acknowledged

Path
DTNRegistry.sol

Function Name
addNodeWorkers()

Description
There's possibility of malicious node addresses associating any node worker addresses with its address using addNodeWorkers() so that, the DTN node which should be associated with the other node worker won't be able to add it.

Recommendation
The solution for this would be for node workers themselves to accept/confirm the in-contract invite created by the DTN node.
Or to allow node workers the option to disassociate themselves with the node, So if any node associates any node worker that is not a part of their network in reality, the node worker will have the right to disassociate itself with that node.

Okto Team's Comment
It does not create a problem. DTNs won't do it. Suppose DTN1 was going to associate nodeworker1 as its worker. Now DTN2 frontruns this transaction and adds nodeworker1 as its own node worker. In this case DTN1 can wrongfully terminate the jobs assigned to DTN2 going forward, as DTN1 will still be holding private keys of nodeworker1 which is now a part of DTN2 worker.

17. The whole reputation score is not subtracted from totalReputationScore

Resolved

Path

DTNRegistry.sol

Function Name

unStake()

Description

In unStake() node's whole reputation score is not subtracted from totalReputationScore. because on L249 the reputationReduction is subtracted and the totalReputationScore can contain the score earned from job termination.

In unStake() node's whole reputation score is not subtracted from totalReputationScore. because on L249 the reputationReduction is subtracted and the totalReputationScore can contain the score earned from job termination.

Recommendation

Consider subtracting dtn.reputationScore from the totalReputationScore.

18. DTN node can utilize flashloan to influence its reputationScore

Acknowledged

Path
JobManager.sol, DTNRegistry.sol

Function Name
initiateJob(), stakeAndRegisterNode(), unStake(), selectNode()

Description
There's possibility of malicious node addresses associating any node worker addresses with its address using addNodeWorkers() so that, the DTN node which should be associated with the other node worker won't be able to add it.

Example: In this case, it is assumed that the node is a contract address with code and can execute calls.

1. A node will take a big flashloan and stake the amount with stakeAndRegisterNode() before initiateJob() executes
2. It will terminate the already assigned job to itself and its weighted score will be updated to include its newly added reputationScore.
3. Then initiateJob() will be executed - because of the greater index range there are more chances of getting selected (as there are more chances of random index ending inside the bigger range than that of low range)
4. Node uses unStake() in the same transaction to unstake the amount that it used to influence the score.
5. Node pays the flashloan in the same transaction.

This process can be repeated to influence the selection process at the expense of flashloan fees.

Recommendation
Use other functionality for node selection which cannot be influenced.

19. Parameters can be replaced by msg.sender

Resolved

Path
DTNRegistry.sol

Function Name
addNodeWorkers(), removeNodeWorker()

Description
_nodeAddress parameter can be removed in both addNodeWorkers() and removeNodeWorker(). and _nodeAddress will be replaced by msg.sender. Doing this will also eliminate the requirement of reverting when the _nodeAddress != msg.sender

Recommendation
Remove _nodeAddress and use the msg.sender instead.

20. onJobCreate() and onJobTerminate() are not emitting any events

Resolved

Path
DerivativeBloc.sol, SwapBloc.sol

Function Name
onJobCreate(), onJobTerminate()

Description
onJobCreate() and onJobTerminate() are not emitting events in both DerivativeBloc.sol, SwapBloc.sol.

Recommendation
Verify the logic and ensure both functions are emitting the events

21. Random number generation is predictable and manipulatable upto some extent.

Acknowledged

Path

JobManager.sol, DTNRegistry.sol

Description

Random numbers generated by contracts for randomIndex are predictable. The code creates pseudorandom numbers.

firstly in the JobManager.initiateJob() calculates random number from the hash of encoded values (block.timestamp, block.prevrandao, _jobParameters)

1. block.timestamp can be guessed in the sequence (e.g for next block it can be previousBlockTimestamp+12 or previousBlockTimestamp+(12-n) or
2. previousBlockTimestamp+(12+n) where the n is the number to get different blocks.) block.prevrandao can be derived from the last block.
3. the _jobParameters are known.

And hence the uint derived from the hash of these values would be predictable.

This uint is getting passed to dtnRegistry.selectNode(_randomValue) which calls _selectRandomNode(_randomValue) to select the random node. firstly it calculates the randomIndex with:

```
uint randomIndex = _randomValue % totalReputationScore;
```

This index can also be predicted because totalReputationScore is already known from the last block. This randomIndex is getting used to decide the DTN node based on the weighted score comparison.

The JobManager.initiateJob() uses block.prevrandao for calculating the uint value that is passed to dtnRegistry.selectNode(_randomValue).

The the block.prevrandao can also be influenced by validator by not signing the current epoch number. In this way the prevrandao value would be from the second last slot. The attacker can do this if its favourable to get his own node address selected.

Recommendation

Use services like Chainlink VRF to get random numbers with oracles.

Okto Team's Comment

Milestone 2 has round-robin selection to avoid this.

22. Unsupported opcode

Resolved

Path

JobManager.sol

Description

initiateJob() uses block.prevrandao opcode for calculating the pseudo random value passed to DTNRegistry.selectNode(). Some chains like polygon zkEVM doesn't fully support block.prevrandao opcode. the block.prevrandao will return 0 instead of random value. So on *L159* the pseudo random value getting passed to DTNRegistry.selectNode() would be calculated with only block.timestamp and _jobParameters and the static value of block.prevrandao which would be 0.

Reference

https://docs.polygon.technology/zkEVM/architecture/protocol/etrog-upgrade/#supported-opcodes

Recommendation

Use services like Chainlink VRF to get random numbers with oracles.

23. Verify and change the function names

Resolved

Path
EVMRawTransactionBloc.sol

Function Name
encodeParams(), decodeParams()

Description
The contract has encodeParams() and decodeParams() for encoding and decoding the params. In other bloc contracts these functions are named as encode() and decode() even though these functions are not a part of IBloc interface which will be used as a standard for other or custom blocs, we recommend verifying the naming convention to avoid any confusion.

Recommendation
Change the function names to encode() and decode() as in the other bloc contracts.

24. Lacks the support for short calldata

Resolved

Path
EVMRawTransactionBloc.sol

Function Name
validate()

Description
validate() checks if the params.data.length is less than 32 then it reverts. but there's a possible case depending on the transaction where the calldata can be less than 32 bytes.

E.g a transaction to call setValue() function does not take any parameter. so the calldata would be only 4 bytes function selector. which is less than 32 bytes. so the validate will revert in this case.

(This also needs to be noted that this 4 bytes values can be entered as 32 bytes value with other bytes as empty bytes to pass this function check e.g 0x4fa519fa00000000000000000000000000000000000000000000000000000000 which shows that the overall impact depends on the usage)

Consider verifying the intended functionality.

Recommendation
Consider allowing data with less than 32 length depending on the requirement.

25. Update the weighted score while staking

Resolved

Path
DTNRegistry.sol

Function Name
stakeAndRegisterNode()

Description
The weighted score is updated in updateNodeReputationScore() when someone terminates the job and in the unStake() function. However, if the business logic requires updating the weighted score on every deposit call, the _updateWeightedScore() can be called in stakeAndRegisterNode().

Recommendation
Update the stakeAndRegisterNode() to call _updateWeightedScore().

Informational Issues

26. Unused event

Resolved

Path
JobManager.sol, DTNRegistry.sol

Description
Following events are unused:
InvalidNodeAddress() in JobManager.sol
ZeroValueNotAllowed() in DTNRegistry.sol

Recommendation
Remove or use the unused events.

27. The code needs to be updated according to the GSN functionality

Resolved

Path
JobManager.sol

Function Name
initiateJob()

Description
Currently the GSN service's functionality is not clear. But some functions like initiateJob() need to be updated according to the intended/required functionality.

E.g If gsnData.gsnToken, gsnData.maxAmount , gsnData.networkId needs to be checked to have the same length can be decided based on the requirement and GSN usage.

Recommendation
Update the initiateJob() functionality according to requirement.

28. Remove the comment

Resolved

Path
JobManager.sol

Function Name
terminateJob()

Description
Remove the comment on *L235* // if (msg.sender != _currentJob.assignedDTN) revert UnauthorizedNode(); as currently the code checks if the worker/msg.sender is not associated with the DTN node then it reverts.

Recommendation
Ensure the commented code is redundant now. And remove the comment to avoid confusion.

29. Malicious jobs can harm the reputation of DTN node

Acknowledged

Path
JobManager.sol

Description
Users can create malicious jobs that would be initiated by authorized job initiators. Suppose DTNs nodes directly execute the job/transaction and the execution fails. While terminating the job with the FAILED status the reputation score would be updated accordingly which can decrease the reputation.

Hence it is required that the selected DTNs should verify that messages are accurate and will be executed correctly on the destination chains.

(This issue cannot be fixed on smart contract level and hence can be acknowledged after taking note.)

Recommendation
Consider taking necessary steps according to suggestions while designing DTNs

Okto Team's Comment
It will be handled in later milestones.

30. Redundant check

Resolved

Path
JobManager.sol

Description
contractIsInitialized() checks for multiple address to be non zero (uses ||) but if any address from networkRegistry, dtnRegistry, blocRegistry, accountsFactory is non zero then other would also be a non zero address as the initialize() sets all four addresses.

Recommendation
Consider checking only 1 address to be non zero instead of all four addresses. The similar changes can be also made to different contacts.

31. Possibility of selecting similar nodes in initiateJob()

Acknowledged

Path

JobManager.sol

Description

If more than one initiateJob() calls are executed in the one block and the _jobParameters is equal for multiple jobs then the selected DTN node would be same for the all initiated jobs(with same _jobParameters) in that block as long as totalReputationScore (which is used to calculate randomIndex) and weightedScores array (getting used to compare with randomIndex to decide the node in _selectRandomNode()) is not getting updated with dtnRegistry.updateNodeReputationScore() and dtnRegistry.selectNode() respectively.

This happens because the way node gets selected is: While initiating the job with initiateJob() it calls dtnRegistry.selectNode() with the uint derived from the hash of these encoded values block.timestamp, block.prevrandao, _jobParameters. Because all the transactions are getting executed in the same block the resulting value for all the jobs would be the same (assuming the _jobParameters is the same for every job).

Then in dtnRegistry.selectNode() it calls _selectRandomNode() which then calculates the randomIndex with randomIndex = _randomValue % totalReputationScore. And as mentioned, because totalReputationScore and weightedScores are not updated the calculated randomIndex would be same and while comparing with the weightedScores array values will also be the same.

These are some situations when it can happen

1. When multiple jobs are created with the same _jobParameters and these jobs are executed in the same block. Which can happen by coincidence.
2. When any smart contract is used as an initiator to initiate the jobs with batch transactions. Which all will be executed in one block as that would be one atomic transaction.

In this way the selected DTN is the same. we couldn't find any issue with one DTN node getting selected multiple times. But we suggest thinking about this possibility when integrating the DTN functionalities in next milestones where things would be more clear.

Recommendation
Consider thinking about this possibility of one node repeatedly getting selected when multiple jobs with same _jobParameters are executed in one transaction.

Okto Team's Comment
This is expected behavior and round-robin selection will be used in milestone 2.

32. Add getter method for some important variables

Resolved

Path
DTNRegistry.sol

Description
Currently registeredNodeAddresses is a private array and there's no way to get the registeredNodeAddresses values. Making this variable public will create a getter method which will help anyone to see registered addresses with static calls, it can be useful in testing and maybe in some future integrations and will also make registered addresses more transparent for the protocol users. Additionally the weightedScores can also be public in case it will help some offchain services to fetch the values and make it easy for ordinary users to read the current weighted scores.

Recommendation
Make variables public to create a getter method for transparency.

33. Care should be taken while adding node workers

Resolved

Path
DTNRegistry.sol

Function Name
addNodeWorkers()

Description
Currently stakeAndRegisterNode() and addNodeWorkers() allow adding node workers.

while adding node workers with addNodeWorkers() currently it's not getting checked if the specific node worker address is already in the allDTNs[_nodeAddress].dtnWorkers array. Hence it is important to check it before adding node workers to avoid duplicate addresses.
Another solution can be to check the same thing on smart contract level. but it would consume extra gas to iterate through the dtnWorkers array

Recommendation
Consider checking if the node workers addresses that are getting added are already in the allDTNs[_nodeAddress].dtnWorkers.

34. Fix compile time error

Resolved

Description
Compile time issues that should be fixed by changing file names for imported files.

1. Error HH409: Trying to import ../interfaces/IDTNRegistry.sol from src/contracts/JobManager.sol, but it has an incorrect casing.
2. Error HH409: Trying to import ../interfaces/INetworkRegistry.sol from src/contracts/NetworkRegistry.sol, but it has an incorrect casing.

Recommendation
Consider checking and changing the file names.

35. It's Possible To Break The Functionality By Specifying The Wrong Login Method

Acknowledged

Path
AccountsFactory.sol

Function Name
registerUserAccount()

Description
registerUserAccount() Function in the AccountsFactory contract allows any external caller to register a user account by providing various parameters, including _loginMethod. This parameter determines how the user authenticates, with options defined by the LoginMethod enum. However, since _loginMethod is passed as an unverified uint. In case any service uses and relies on the login method information stored in the smart contract for any tasks, It's possible to break the functionality by specifying the wrong login method.

Recommendation
Check required access control is in place.

Okto Team's Comment
It will be handled in later milestones.

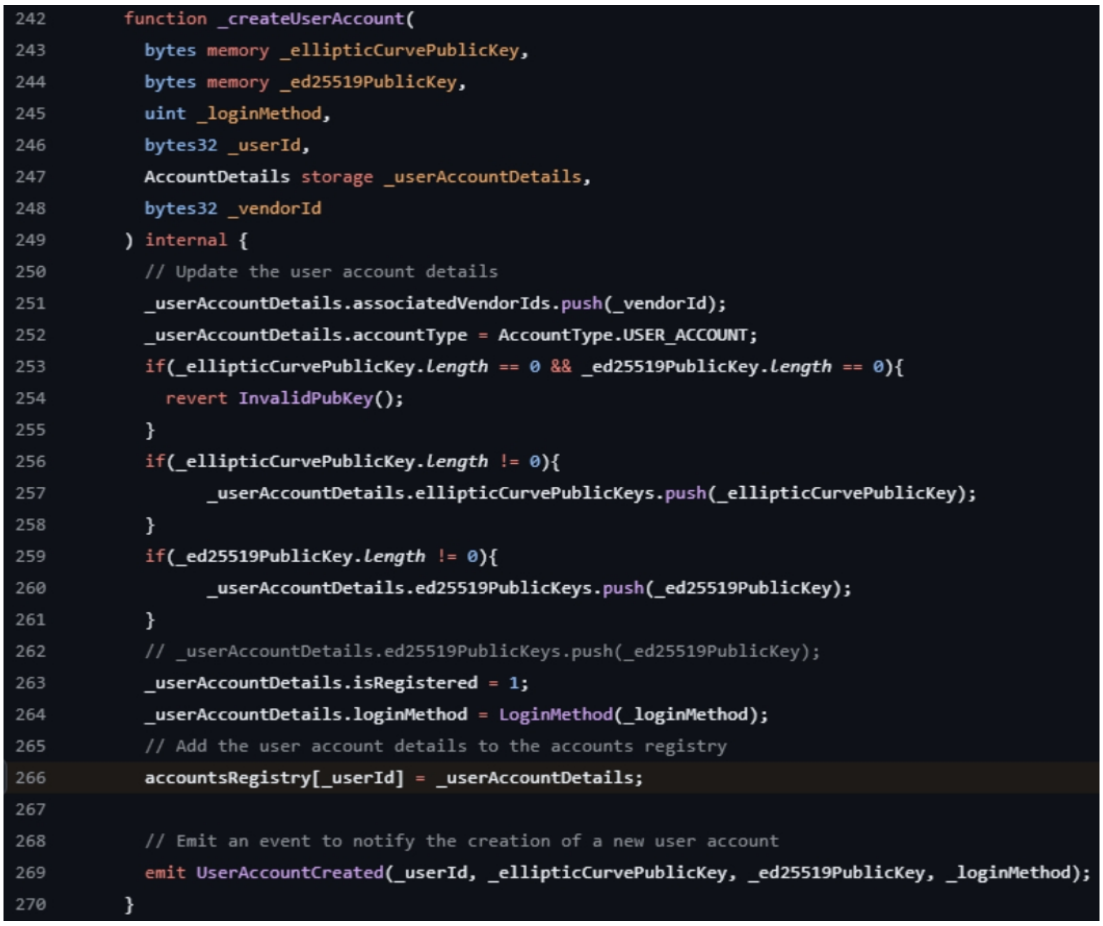
36. Redundant storage assignment in _createUserAccount() function

Resolved

Path
AccountsFactory.sol

Function Name
_createUserAccount()

Description
In _createUserAccount() on L266 there's redundant storage assignment accountsRegistry[_userId] = _userAccountDetails;. This assignment is redundant because in _createUserAccount() the _userAccountDetails is a storage reference that was created in the registerUserAccount() on L127, AccountDetails storage userAccountDetails = accountsRegistry[_userId];. as can be seen on L127 it's for the same _userId from accountsRegistry mapping.



```
242     function _createUserAccount(
243       bytes memory _ellipticCurvePublicKey,
244       bytes memory _ed25519PublicKey,
245       uint _loginMethod,
246       bytes32 _userId,
247       AccountDetails storage _userAccountDetails,
248       bytes32 _vendorId
249     ) internal {
250       // Update the user account details
251       _userAccountDetails.associatedVendorIds.push(_vendorId);
252       _userAccountDetails.accountType = AccountType.USER_ACCOUNT;
253       if(_ellipticCurvePublicKey.length == 0 && _ed25519PublicKey.length == 0){
254         revert InvalidPubKey();
255       }
256       if(_ellipticCurvePublicKey.length != 0){
257             _userAccountDetails.ellipticCurvePublicKeys.push(_ellipticCurvePublicKey);
258       }
259       if(_ed25519PublicKey.length != 0){
260             _userAccountDetails.ed25519PublicKeys.push(_ed25519PublicKey);
261       }
262       // _userAccountDetails.ed25519PublicKeys.push(_ed25519PublicKey);
263       _userAccountDetails.isRegistered = 1;
264       _userAccountDetails.loginMethod = LoginMethod(_loginMethod);
265       // Add the user account details to the accounts registry
266       accountsRegistry[_userId] = _userAccountDetails;
267
268       // Emit an event to notify the creation of a new user account
269       emit UserAccountCreated(_userId, _ellipticCurvePublicKey, _ed25519PublicKey, _loginMethod);
270     }
```

Recommendation
Remove the redundant line accountsRegistry[_userId] = _userAccountDetails;

37. Vendor Cannot Be Registered If Already Registered As a User In registerUserAccount() Function

Resolved

Path

AccountsFactory.sol

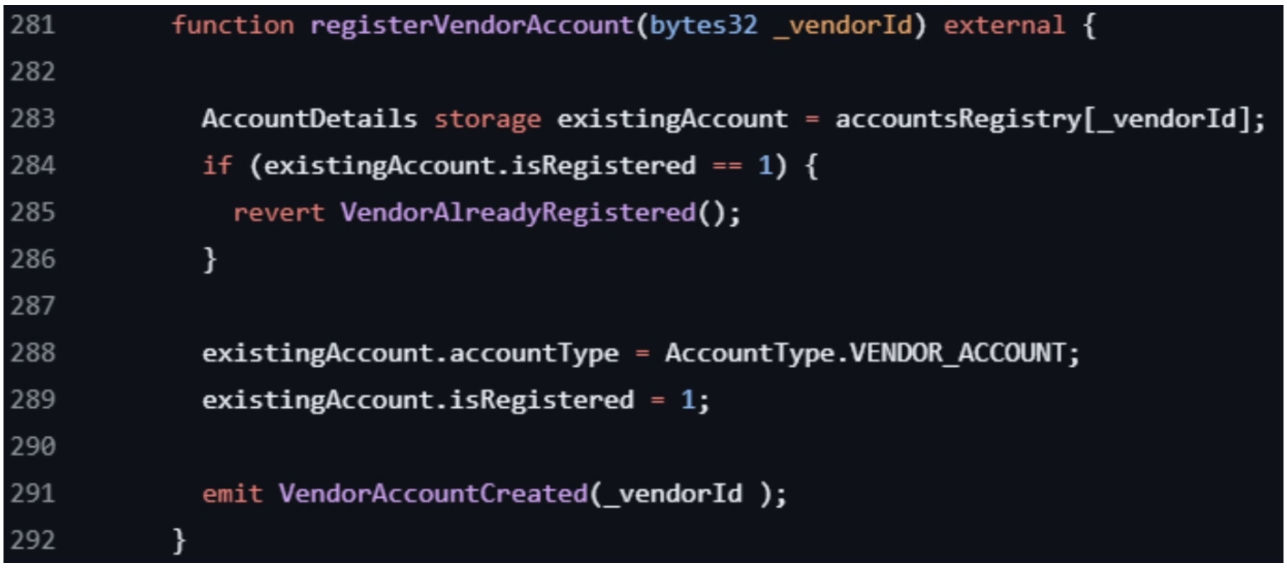
Function Name

registerVendorAccount()

Description

Any user who registers using the registerUserAccount() function will have their isRegistered state updated to 1.

If the same user attempts to register as a vendor, the registration will revert.



```
    function registerVendorAccount(bytes32 _vendorId) external {

      AccountDetails storage existingAccount = accountsRegistry[_vendorId];
      if (existingAccount.isRegistered == 1) {
        revert VendorAlreadyRegistered();
      }

      existingAccount.accountType = AccountType.VENDOR_ACCOUNT;
      existingAccount.isRegistered = 1;

      emit VendorAccountCreated(_vendorId );
    }
```

Recommendation

Consider checking if the functionality matches the requirements.

Okto Team's Comment

It is expected behavior.

38. Length does not matter while pushing _ed25519PublicKey to AccountDetails

Resolved

Path
AccountsFactory.sol

Function Name
_createUserAccount()

Description
In the _createUserAccount function, the _ed25519PublicKey is always pushed to the AccountDetails struct, regardless of whether it is valid or has any length.

If the same user attempts to register as a vendor, the registration will revert.

Recommendation
Remove the L262 _userAccountDetails.ed25519PublicKeys.push(_ed25519PublicKey);

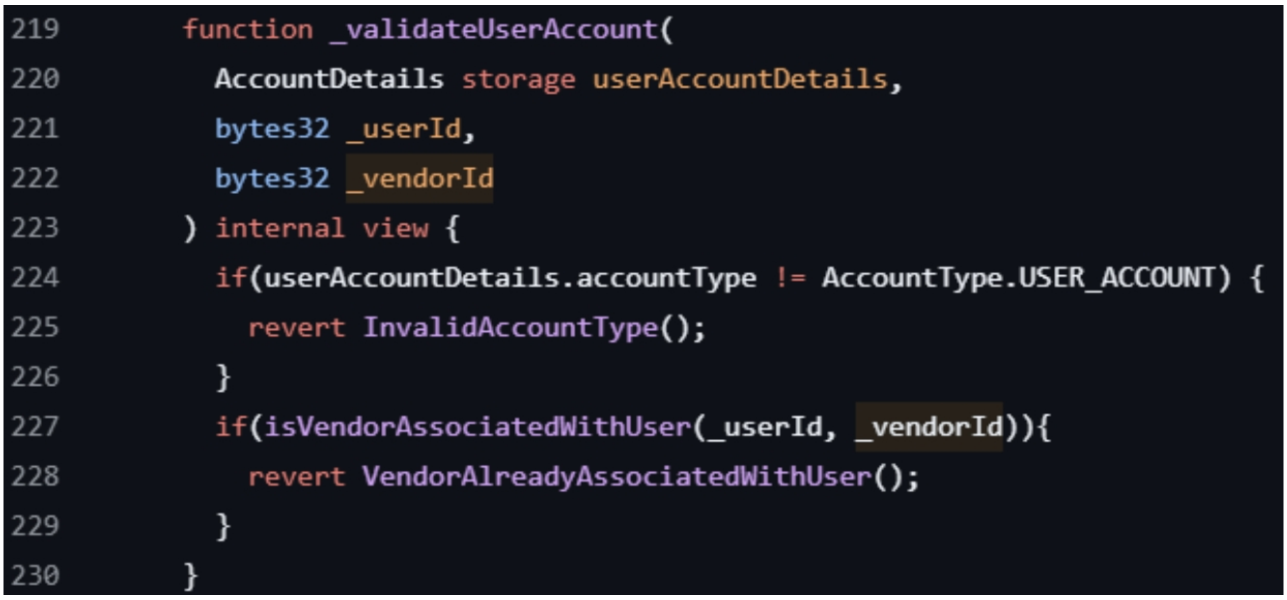
39. Unregistered Users Treated as Valid Accounts

Resolved

Path
AccountsFactory.sol

Function Name
addVendorAccountToUser()

Description
The addVendorAccountToUser() function verifies that the account type of the provided user ID is AccountType.USER_ACCOUNT by calling the _validateUserAccount() function. However, it is important to note that for unregistered IDs, the accountType value would be to 0 by default, which corresponds to USER_ACCOUNT due to Solidity's default enum behavior.



```
function _validateUserAccount(
  AccountDetails storage userAccountDetails,
  bytes32 _userId,
  bytes32 _vendorId
) internal view {
  if(userAccountDetails.accountType != AccountType.USER_ACCOUNT) {
    revert InvalidAccountType();
  }
  if(isVendorAssociatedWithUser(_userId, _vendorId)){
    revert VendorAlreadyAssociatedWithUser();
  }
}
```

As a result, _validateUserAccount() does not revert for unregistered users, which violates the intended requirement of ensuring only registered users can associate vendor accounts.

Recommendation
Make necessary changes.

40. Insufficient Check In addKeyToUserAccount() Function

Resolved

Path
AccountsFactory.sol

Function Name
addKeyToUserAccount()

Description
The _createUserAccount() checks if the length of the public keys is non-zero. However, in addKeyToUserAccount() there's no check for checking this. Hence a similar check can be added.

Recommendation
Add suggested checks.

41. ERC20 import and inheritance is redundant

Resolved

Path

OktoTokenTest.sol

Description

ERC20.sol is getting imported and inherited even when the abstract ERC20Permit already inherits the ERC20. hence the erc20 import and inheritance is redundant and can be removed.

Recommendation

Redundant Import and inheritance can be removed.

42. Verify the functionality matches the logic

Resolved

Path

OktoTokenTest.sol

Description

Currently, the owner can mint an unlimited amount of token supply. verify that if the current functionality matches the business logic.

Recommendation

Verify the functionality matches the logic and make necessary changes.

43. addSupportedNetworks() allows adding duplicate network string

Resolved

Path
NetworkLibrary.sol

Function Name
addSupportedNetworks()

Description
The function fails to check the possibility of a duplicate network string in the supportedNetworks array.

Remediation
Add a check to verify duplicate networks. This way it will prevent pushing duplicates to the allSupportedNetworksByBloc array.

44. Does not revert if the owner tries to remove the network that is not yet added in the bloc but supported by the NetworkRegistry

Resolved

Path
NetworkLibrary.sol

Function Name
removeSupportedNetworks()

Description
When bloc owners intend to remove unknown networks with the removeBlocSupportedNetworks() function, it does not revert but rather sets value to 0 even though unknown networks are expected to have zero value mapped.
For example, A bloc owner attempts to remove “XYZNetwork” with removeBlocSupportedNetworks(“XYZNetwork”), Lets assume at this moment there’s no network by the name XYZNetwork in the bloc. The networklibrary.removeSupportedNetworks() will execute in this case. it will just assign 0 value by using isNetworkSupported[network] = 0; on L78 after checking that the network that the bloc owner is trying to remove is supported by networkRegistry. But in reality there was no network XYZNetwork added to the bloc (even though it was present/supported in the NetworkRegistry).

Remediation
Add a check that verifies which networks are supported by the bloc contracts when removeBlocSupportedNetworks() is invoked, before the execution flow reaches networklibrary.removeSupportedNetworks().

45. Anyone can add keys to user account using addKeyToUserAccount

Resolved

Path
AccountsFactory.sol

Function Name
addKeyToUserAccount()

Description
There is no restriction to this function and for that reason, anyone can add keys to a user account. If users are expected to receive tokens to these blockchain public keys, malicious users can add their own public key. There is a tendency that the original user will lose tokens to these malicious users if the public key is used to send tokens.

Remediation
Add restrictions to this function to only allow account owners to add to their public keys.

46. Check the iterative usage with the same parameters.

Resolved

Path
AccountsFactory.sol

Function Name
addKeyToUserAccount(), addVendorAccountToUser()

Description
Some functions in this contract are callable by anyone. A malicious user can add a vendor multiple times to users account, increasing the length of associatedVendorIds. Malicious users can overpopulate users' accounts with their own public keys. The contract fails to check that a previously added item does not get added again.

Remediation
Ensure that there is a restriction to who can add public keys and vendor ids to users account. Additionally a check can be added to verify that an item does not get added twice.

47. Memory variable can be used instead of storage reference to save gas usage

Resolved

Path
JobManager.sol

Function Name
terminateJob()

Description
On L250 the job status is getting set with _currentJob.status = JobState(_status); on L255 the JobTerminated() event is emitted which also emits the status of job. this status is getting passed as storage reference _currentJob.status. Accessing storage is costly as compared to accessing memory.

Remediation
Instead of passing the storage reference, memory variable _status can be passed to save gas.

General Notes

The audit team is unable to verify some things as there are some remaining milestones for this project. We recommend checking, verifying the intentional functionality and keeping the implications in mind while working on next milestones.

1. Anyone can register the user and add the vendor as associated with that user address using registerUserAccount(). We understand that allowing anyone to add the user can create a more simple flow where it won't cost the fees to original users, still we recommend verifying that this is intentional functionality. In some cases like when offchain service will try to register the users directly for the first time it can fail if the user is already registered by any other party.
2. Anyone can register the vendor using registerVendorAccount(). Check if it's intentional. In cases like when offchain service will try to register the vendor directly for the first time it can fail if the user is already registered by any other party.

Functional Tests

Some of the tests performed are mentioned below:

JobManager.sol

- ✓ Authorized initiator should be able to initiate the job
- ✓ Associated worker should be able to terminate the job
- ✓ Updating the GSN params format with updateGsnParamsFormat() should not affect other functionality
- ✓ Terminating already terminated jobs should not be possible
- ✓ should not be able to initiate an already executed job

DTNRegistry.sol

- ✓ addNodeWorkers() should associate the added node workers with the node
- ✓ Anyone should not be able to overwrite the mapped values for node worker
- ✓ Anyone should not be able to stake and register node
- ✓ Should handle the precision loss while calculating reputationScore
- ✓ The user should not be able to overpopulate the registeredNodeAddresses[] array with same address
- ✓ Executing select node logic multiple time in the one block
- ✓ Should not be able to add duplicate node workers with addNodeWorkers()

NetworkLibrary.sol

- ✓ Should not be able to add duplicate network
- ✓ Should revert when the owner tries to remove a network not added to the bloc but supported by NetworkRegistry
- ✓ Should be able to remove the network from the block which is currently not supported by the NetworkRegistry

Functional Tests

AccountsFactory.sol

- ✓ Should not be able to add multiple keys to a user account
- ✓ Should revert if trying to register an already registered user

BlockRegistry.sol

- ✓ Should be able to register the bloc
- ✓ Should be able to transfer the bloc ownership
- ✓ checkBlocActive() should return the expected bloc status

EVMRawTransactionBloc.sol

- ✓ validate() should support the short data

NFTMintBloc.sol

- ✓ validate() reverts if the metadata is empty

SwapBloc.sol

- ✓ Validate() allows 0/empty slippage amount

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Okto milestone 1. We performed our audit according to the procedure described above.

Some issues of high, medium , low and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Okto milestone 1. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Okto milestone 1. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Okto Team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

January 2025

For

okto

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com