HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SubQuery PTE. LTD.
Date: July 7th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for SubQuery PTE. LTD.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://subquery.network
Timeline	06.06.2022 - 07.07.2022
Changelog	14.06.2022 - Initial Review 07.07.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by SubQuery PTE. LTD. (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/subquery/contracts
Commit:
b511941610300f0d42d3d008625a092b03e0f4e5
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://static.subquery.network/whitepaper.pdf

Type: Functional requirements
Link: https://github.com/subquery/contract-audit-batch1/tree/draft/docs

Type: Technical description
Link: https://github.com/subquery/contracts/blob/main/package.json

Hardhat tests: Yes
Contracts:
File: ./contracts/ProxyAdmin.sol
SHA3: 18099a93f881d3321d946f697e825d2256b5d90bb044dfe197190e26

File: ./contracts/QueryRegistry.sol
SHA3: 6f8c959f17d888ccf0c8c8b0c41fbd780905d59ecbb69a6d3cc25c75

File: ./contracts/Settings.sol
SHA3: 27bec60bd81e88348d182927156eb695c50412dd31b3dac8906f2b99

File: ./contracts/IndexerRegistry.sol
SHA3: 5d386732279c5e9d53cd68a01e3b6eb46bc073738ecde932bac1858a

File: ./contracts/AdminUpgradeabilityProxy.sol
SHA3: d559369da2199ac3405a77773f4091422c91909e3547480d51c609b4

File: ./contracts/ClosedServiceAgreement.sol
SHA3: 396e531cd2b9f65fe6984a832227d52e9e30eb12aed77dcadaea10d7

File: ./contracts/Staking.sol
SHA3: 5486c0882af52ee1cf5695c45d77844434c0bdf92c2ad53fcd328115

File: ./contracts/RewardsDistributer.sol
SHA3: 311b6be8a4f329e857309d6964be51464267c4979ac7c682ad0323d1

File: ./contracts/SQToken.sol
SHA3: 557c68b070cd8f8a325a4c391029b3b0d39806f02b99bfe80635df54

File: ./contracts/Constants.sol
SHA3: a2efa6cb5be3763bb0f1baa7f9e5b6ddb7bb8867e7253fedf27e007c

File: ./contracts/PurchaseOfferMarket.sol
SHA3: 100398fb309d73de81e2f42f3844949337cd4a71fddb7a103bb73cf5

File: ./contracts/ServiceAgreementRegistry.sol
SHA3: 9817c8d282550b13c8fd81b2255b2a458e1a50b5c3857b7b705eb3fb

File: ./contracts/PlanManager.sol
SHA3: 3a05dd481344204dd682f23dd459377602895c4714068c59d6c66b5c

File: ./contracts/InflationController.sol
SHA3: 4dccb87d253f9954e43aa2ae7495b314829e92d6e951a4b0a9155d7f

File: ./contracts/MathUtil.sol
SHA3: ed82d2c60d8d9d857a925ed696544db638f07b1dd67c745c17028e8b

File: ./contracts/EraManager.sol
SHA3: 8b256fffe4766148e5c4a1fb1545e4316bb7b9264cf683c99fb1dd70

Second review scope

Repository:
https://github.com/subquery/contracts

Commit:
753e141085722fd026d76887c5b7127a5e99aaab

Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://static.subquery.network/whitepaper.pdf

Type: Functional requirements
Link: https://github.com/subquery/contract-audit-batch1/tree/draft/docs

Type: Technical description
Link: https://github.com/subquery/contracts/blob/main/package.json

Hardhat tests: Yes

Contracts:

File: ./ProxyAdmin.sol
SHA3: 5037a74c708c8852d6f5014fc91b15194551f623d8b60c5984c69ca27e9998504bdfc

File: ./QueryRegistry.sol
SHA3: 61e6548758439740c237ff05026efb09e1fbf4a863f822294478222c

File: ./Settings.sol
SHA3: fe2a881ad3d152eff2eeb3b21165f9e8929f9bdbcc6454352d0ebd2e

File: ./IndexerRegistry.sol
SHA3: 8c39e3856f8ce8173694c6dc151fe0e6b19891905193124f37ed1cb0

File: ./AdminUpgradeabilityProxy.sol
SHA3: b211e9e1766f7cc982c15ce8d90166a1dbd3a2dddf698929dd258b02

File: ./ClosedServiceAgreement.sol
SHA3: 12306a34add6b000f37d0b09d831d072d2b994593f9557254becb3f5

File: ./Staking.sol
SHA3: e9970868894133c5378a34201441cd102a0320e34585f30f7e6a368f

File: ./RewardsDistributer.sol
SHA3: cb2b5251f377145fb1fc8371de50a0977089487659cc210671f9f24d

```
File: ./SQToken.sol
SHA3: 706cd8bd853301950de56949bab863502e3ea57f29e91990ddf91463

File: ./Constants.sol
SHA3: 21bef0d3e1474bb4beb9338c2915b6497376f96fed9b2f62ef038309

File: ./PurchaseOfferMarket.sol
SHA3: 66f490f25c470eb6781209fc2a3a25655c330a58d23af698aa69008f

File: ./ServiceAgreementRegistry.sol
SHA3: 30f771a47b2fa89a66343683c36c1b4e5f2b4eea2669d82b1176176e

File: ./PlanManager.sol
SHA3: f9f811a71a725bc83975f5187f28428f5d316c1810dd13ec862824ae

File: ./InflationController.sol
SHA3: 1e5d5816d45c9d1a5490555da4641aa77673d05cfb801c7b3c326f03

File: ./MathUtil.sol
SHA3: 403ec24a3b247b60f08a91418a04b6fdbdc68ea1298113df115de15b

File: ./EraManager.sol
SHA3: 2317bdaaaa5a84a53bdc04a2aae6dd92c832c5a12ba890549aa43a50
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. The Customer provided good functional requirements and a clear technical description.

Code quality

The total CodeQuality score is **8** out of **10**. Code is written carefully, the code has significant but not full test coverage.

Architecture quality

The architecture quality score is **10** out of **10**. The logic is carefully separated into several contracts. The contract system is clear.

Security score

As a result of the audit, security engineers found **2** medium and **2** low severity issues. The security score is **8** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.4**.

1
2
3
4
5
6
7
8
9
10
You are here

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race	SWC-114	Race Conditions and Transactions Order	Passed

Conditions		Dependency should not be possible.	
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
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.	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Not Relevant
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.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of	Failed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

SubQuery Network is a mixed-purpose contract system which includes scope of the audit. In the scope were audited token contract and staking system:

- *EraManager* - manager of custom time periods called an era. Each era is at least constant duration but could be longer if the contract does not interact for some time. Features:
    - start new era
    - get current era
    - update minimal era duration (owner only)
- *IndexerRegistry* - manager of registered indexers who can manage staked assets. Features:
    - register/unregister self as indexer
    - set/remove controller account for indexer (indexer only)
- *InflationController* - manager of total amount tokens, mints tokens to special destination according to setted inflation rate. Features:
    - set inflation rate (owner only)
    - mint inflated tokens
- *PlanManager* - manager of plan templates and actual staking plans. Features:
    - create/remove plan template (owner only)
    - create/remove plan for consumers (indexer only)
    - accept plan entering into it
- *PurchaseOfferMarket* - platform for consumers to create offers acceptable to indexers. Features:
    - create/remove offer for indexers
    - accept offer starting indexing it (indexer only)
- *QueryRegistry* - platform for consumers to place projects that indexers may take into account. Features:
    - create query
    - remove query (if not expired, penalty may be taken)
    - start/stop indexing query (indexer only)
- *RewardsDistributer* - supporting contract for paying rewards.
- *ServiceAgreementRegistry* - manager of agreements between indexers and consumers. Features:
    - establishing agreement
    - renewing agreement
    - removing ended agreement
- *Settings* - supporting contract that manages all other contract addresses. Its address is stored in all other contracts.
- *SQToken* - simple ERC-20 token. Features:
    - minting by specified minter (InflationController)
    - burning owned tokens
    - name: SubQueryToken
    - symbol: SQT

- *Staking* - staking contract based on service agreements between indexer and customer. Features:
    - stake assets (indexer only)
    - delegate/redelegate assets
    - withdraw assets (commission is taken)

Privileged roles

EraManager:

- *Owner* - may update the duration of the era period

IndexerRegistry:

- *Owner* - may set minimum staking amount and settings contract

InflationController:

- *Owner* - may set inflation rate and inflation destination

PlanManager:

- *Owner* - may set indexer plan limit, create new plan template

ProxyAdmin:

- *Owner* - may change proxy admin and upgrade implementation

PurchaseOfferMarket:

- *Owner* - may set penalty rate, penalty destination
- *Indexer* - may accept purchase offer

RewardsDistributer:

- *Owner* - may set settings contract

ServiceAgreementRegistry:

- *Owner* - may set settings contract and staking threshold

Settings:

- *Owner* - may set contracts for: era manager, staking, token, indexer registry, service agreement registry, rewards distributer, inflation controller

SQToken:

- *Owner* - may set minter and transfer ownership
- *Minter* - may mint an unlimited amount of tokens

Staking:

- *Owner* - may set settings contract, staking lock period, unbound fee

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

1. Inconsistent state

The *createPlan* function increments the *planCount* value for a msg.sender. Though, this value is not decremented when plans are removed.

The contract state can be inconsistent.

File: ./contracts/PlanManager.sol

Function: removePlan

Recommendation: decrement the *planCount* value when removing old plans.

Status: Fixed (second review)

🟩🟩 Medium

1. Possible rudiment code

According to comments, the functions and the mapping should do something. However, the mapping is never used in the contract system, so possibly it is a rudiment from old logic implementation.

Redundant state variables and functions lead to unnecessary gas usage during the deployment of the contracts.

File: ./contracts/ServiceAgreementRegistry.sol

Functions: removeUser, addUser

Mapping: consumerAuthAllows

Recommendation: remove logic related to the *consumerAuthAllows* state variable or use the state variable in the contract logic.

Status: Mitigated (the mapping is used offchain)

2. Missing check for constructor parameter

Some checks are provided on updating state variables by the owner, but initial values of the variables are not checked in the constructor.

File: ./contracts/EraManager.sol

Parameter: _eraPeriod (should be > 0)

File: ./contracts/InflationController.sol

Parameter: _inflationRate (should be < *PER_MILL*)

File: ./contracts/PurchaseOfferMarket.sol

Parameter: _penaltyRate (should be < *PER_MILL*)

Recommendation: check initial values to be in bounds.

Status: Fixed (second review)

3. **Missing check for return value**

   Return value should be taken into account. Return value of ERC-20 *transfer* and *transferFrom* functions should be considered too.

   File: ./contracts/PlanManager.sol

   Function: acceptPlan

   File: ./contracts/PurchaseOfferMarket.sol

   Functions: createPurchaseOffer, cancelPurchaseOffer, acceptPurchaseOffer

   Recommendation: check return values in all cases.

   Status: Fixed (second review)

4. **Violating Check-Effect-Interaction pattern**

   The pattern should not be violated.

   It may cause problems in future development. It lowers the fault tolerance of the whole system.

   File: ./contracts/IndexerRegistry.sol

   Functions: registerIndexer, unregisterIndexer

   File: ./contracts/InflationController.sol

   Function: mintInflatedTokens

   File: ./contracts/PurchaseOfferMarket.sol

   Functions: createPurchaseOffer, acceptPurchaseOffer

   File: ./contracts/QueryRegistry.sol

   Function: stopIndexing

   File: ./contracts/RewardsDistributer.sol

   Functions: distributeRewards, _updateTotalStakingAmount, _claim

   Recommendation: provide all interactions even with trusted contracts after all state variables change.

   Status: Fixed (second review)

5. **Shadowing state variables**

Shadowing state variables may cause issues in future development.

File: ./contracts/QueryRegistry.sol

Functions: updateQueryProjectMetadata, updateDeployment

Recommendation: use unique variable names.

Status: Fixed (second review)

6. **Missing validation**

The function proceed execution even if *passedTime* is 0.

File: ./contracts/InflationController.sol

Function: mintInflatedTokens

Recommendation: do not proceed execution if *passedTime* is 0.

Status: Fixed (second review)

7. **Possible Gas limit exceeding**

The Gas limit may be exceeded in cycles whose iteration count does not depend on parameters and is not bounded to a specific number.

In order to fix it, page navigation through the data set may be implemented, or data size may be bounded to a reasonable value.

File: ./contracts/PlanManager.sol

Functions: templates, indexerPlans

File: ./contracts/RewardsDistributer.sol

Function: getPendingStakers

File: ./contracts/ServiceAgreementRegistry.sol

Function: clearAllEndedAgreements

File: ./contracts/Staking.sol

Function: getUnbondingAmounts

Recommendation: make mentioned loop size predictable.

Status: Reported

8. **Missing check for return value**

Return value should be taken into account. Return value of ERC-20 *transfer* and *transferFrom* functions should be considered too.

File: ./contracts/ServiceAgreementRegistry.sol

Function: renewAgreement

Recommendation: check return values in all cases.

Status: Reported

Low

1. Missing explicit visibility levels

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

File: ./contracts/ClosedServiceAgreement.sol

Variable: settings

File: ./contracts/QueryRegistry.sol

Variables: offlineCalcThreshold, deploymentIds

File: ./contracts/RewardsDistributer.sol

Variables: info, pendingStakers, pendingStakerNos, delegation, pendingStakeChangeLength, lastSettledEra, totalStakingAmount, pendingCommissionRateChange

File: ./contracts/Staking.sol

Variables: totalStakingAmount, unbondingAmount, delegation, stakingIndexerLengths

Recommendation: explicitly define visibility for all state variables.

Status: Fixed (second review)

2. Floating pragma

The contracts use floating pragma ^0.8.10.

Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (second review)

3. Tautology check

uint256 is always >= 0, it is not necessary to check it.

File: ./contracts/InflationController.sol

Function: setInflationRate

Recommendation: remove tautology.

Status: Fixed (second review)

4. Comparisons of boolean values

Boolean constants can be used directly and do not need to be compared with *true* or *false*.

File: ./contracts/QueryRegistry.sol

Functions: createQueryProject, updateDeployment

File: ./contracts/PlanManager.sol

Functions: createPlan, removePlan, acceptPlan

Recommendation: remove the equality to the boolean constant.

Status: Fixed (second review)

5. **Functions that could be declared as external**

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

File: ./contracts/ProxyAdmin.sol

Functions: getProxyImplementation, getProxyAdmin, changeProxyAdmin, upgrade, upgradeAndCall

File: ./contracts/ServiceAgreementRegistry.sol

Functions: clearAllEndedAgreements, serviceAgreementExpired, getIndexerDeploymentSaLength

File: ./contracts/Staking.sol

Functions: setInitialCommissionRate, setCommissionRate

Recommendation: use the *external* attribute for functions never called from the contract.

Status: Fixed (second review)

6. **Missing zero address validation**

Address parameters are used without checking against the possibility of being 0x0.

This can lead to unwanted external calls to 0x0.

File: ./contracts/Settings.sol

Functions: setAllAddresses, setSQToken, setEraManager, setServiceAgreementRegistry, setRewardsDistributer, setInflationController

Recommendation: implement zero address validations.

Status: Fixed (second review)

7. **Unnecessary import statements**

Unnecessary import statements may make code unclear.

File: ./contracts/ClosedServiceAgreement.sol

Import:
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol
./interfaces/ISettings.sol
./interfaces/IIndexerRegistry.sol

File: ./contracts/EraManager.sol

Import: @openzeppelin/contracts/access/Ownable.sol

File: ./contracts/IndexerRegistry.sol

Imports:
./interfaces/IIndexerRegistry.sol,
./interfaces/IRewardsDistributer.sol

File: ./contracts/PurchaseOfferMarket.sol

Import: @openzeppelin/contracts/access/Ownable.sol

File: ./contracts/ServiceAgreementRegistry.sol

Import: ./interfaces/IIndexerRegistry.sol

Recommendation: remove unnecessary import statements.

Status: Fixed (second review)

8. Missing zero address validation

Address parameters are used without checking against the possibility of being 0x0.

This can lead to unwanted external calls to 0x0.

File: ./contracts/Staking.sol

Function: setSettings

Recommendation: implement zero address validations.

Status: Reported

9. Unnecessary parameter

According to the current realization, the parameter should always be equal to *msg.sender*, so the check and parameter may be removed to save Gas.

File: ./contracts/ServiceAgreementRegistry.sol

Functions: addUser, removeUser

Parameter: consumer

Recommendation: remove the parameter and the check.

Status: Reported

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.