Security Research Labs

Peaq Baseline Security Assurance

Hacking assessment report

14th of February 2025, v1.0

Prepared for:
Polkadot Assurance Legion &
Peaq Networks

Content

Disclaimer 4
All Rights Reserved 4
Timeline 5
1 Executive Summary 6
1.1 Engagement Overview 6
1.2 Observations and Risk 6
1.3 Recommendations 6
2 Evolution suggestions 7
2.1 Engage in an Economic audit for the Tokenomics mechanism design 7
2.2 Improve the documentation and inline comments 7
2.3 Regular updates 7
2.4 Regular code review and continuous fuzz testing. 7
2.5 Launch a bug bounty program 7
3 Motivation and scope 8
4 Methodology 9
4.1 Security design coverage check. 9
4.2 Implementation check 9
4.3 Remediation support 9
5 Findings summary 11
5.1 Risk profile 11
5.2 Issue summary 12
6 Detailed findings 13
6.1 S3-43: ExistentialDeposit is configured to 0 13
6.2 S3-40: Incorrectly on_finalize weights might lead to denial-of-service attacks 14
6.3 S3-39: Collator can drain delegator's rewards by manipulation commission rate 15
6.4 S3-36: Missing trait-in-use checks in _validateTraitOwnership enables infinite minting 16
6.5 S2-37: Single trait approval for NFT minting limits usability and efficiency 17
6.6 S2-35: Trait token may be reused to satisfy multiple trait slots 18
6.7 S2-18: Permissive `GasLimitStorageGrowthRatio` leads to excessive storage growth 19
6.8 S2-16: Incorrect benchmarks for `pallet_evm` 20
6.9 S1-42: Unsafe arithmetic can halt collator payouts 21
6.10 S1-41: Lack of weight tracking in note_author() hook 22
6.11 S1-32: Missing sanity checks for traitContract address 23
6.12 S1-31: Single-step ownership transferal 24
6.13 S1-15: Incorrect topic selector for `RemoveAttribute` event submission 25
6.14 S0-38: Gas optimizations and logic efficiency 26
6.15 S0-33: Missing events in SolarSeekers and SolarSeekersTraits contracts 27

6.16 S0-19: Missing size checkings could lead to unnecessary gas cost 28
6.17 S0-17: Incorrect fields names in reverted function backtraces..29
Bibliography 30

Version:	Final, v1.0	
Prepared For:	Peaq Network LTD	
Date:	14th February 2024	
Prepared By:	Kevin Valerio Aarnav Bos Cayo Fletcher-Smith	kevin@srlabs.de aarnav@srlabs.de cayo@srlabs.de

Timeline

The Peaq Network's source code has undergone an initial baseline audit for NFT Smart Contracts, Tokenomics and Precompiles started in October 2024 by Security Research Labs. The timeline of the audit and the components audited as shown in Table 1.

Audited On	Components
October 2024	Precompiles
November-December 2024	Tokenomics & NFT Smart Contract

Table 1: Security assurance timeline

1 Executive Summary

1.1 Engagement Overview

This work describes the results of the security audit conducted from October to December 2024 for the following Peaq Network components: NFT Smart Contracts, Tokenomics and Precompiles. Security Research Labs is a consulting firm that has been providing specialized audit services in the Polkadot and Substrate ecosystem since 2019.

During this assessment, the Peaq Network team provided access to relevant documentation, code repository and supported the research team effectively. The implementation of Peaq Network's source code was verified to assure that the business logic of the product is resilient to hacking and abuse.

Security Research Labs has conducted continual comprehensive security assurance for Peaq Network in partnership with Polkadot Assurance Legion (PAL) since our baseline audit in March 2024. The security assurance mainly focused on the following areas of scrutiny along with its subcomponents as follows:

- **Smart Contracts**. Focused on the correct integration of the ERC-721 standard, including token minting controls; derived token relationships and safe transferability; access control implementation; and logic efficiency optimizations.
- **Tokenomics**. Focused on collator centric mechanisms such as the collator selection processes; collator slashing mechanisms; and collator stake-based rewards and token distribution mechanisms.
- **Precompiles**. Focused on Solidity precompile implementation and the correct configuration of the Ethereum execution environment. This included Substrate weight to runtime gas conversion, such as MBIP-5; accurate accounting of precompile storage access operations; correct implementation of precompile business logic; safe integration of pallet-evm into the runtime configuration.

Our testing approach combined manual code review and static analysis techniques utilizing both automated tools and manual analysis of the generated results. We prioritized the review of critical functionalities and the execution of thorough security tests to ensure the robustness of Peaq's platform. Throughout the review process, the audit team collaborated closely with Peaq developers, utilizing full access to source code, documentation, and the development team to perform a rigorous assessment.

1.2 Observations and Risk

The research team identified 4 High, 4 Medium, 5 Low and 4 Info level severity issues (a total of 17 issues), which concerned mostly fee calculations, inaccurate weights, incorrect benchmarking, usage of unsafe arithmetic and bugs in business logic implementation. Peaq Network has acknowledged all the reported issues and in cooperation with the auditors, remediated a subset of identified issues.

1.3 Recommendations

Security Research Labs recommends increasing the test cases for the NFT Smart Contract to validate logic correctness and to improve the edge cases in the implementation. This will facilitate better reasoning about the code, testing for logical and functional correctness of the implementation during the security audit. It is also important to consider improving the documentation of the NFT Smart Contract to reflect the current design through a dedicated documentation page and with in-line comments on the implementation of these contracts. We also recommend engaging in continuous security audits as the codebase evolves, fixing the remaining open issues from this audit, and seeking remediation support when these issues are fixed as not to introduce additional bug into the codebase.

2 Evolution suggestions

2.1 Engage in an Economic audit for the Tokenomics mechanism design

Although Security Research Labs has some knowledge of economic attacks, our primary goal during the engagement was to find logic vulnerabilities through code assurance. Correctness of solutions regarding economic modelling and reward mechanisms cannot be verified in security audits. The economic parameters of the NFT Smart Contract must be carefully reviewed before launch to avoid economic attacks on the network. Some formulae with economic significance to the network operation and monetary system – including snapshotting, reward calculation, and distribution must be verified for correctness. While the implementation of these formulae was audited for arithmetic bugs, the correctness and assumptions considered in the mechanism of these formulae derivation should be verified through a separate exhaustive economic audit.

2.2 Improve the documentation and inline comments

The NFT Smart Contract has outdated documentation [1], and the code is very lightly commented without including explanations on its rationale and its interaction with the other contracts. This could lead to misunderstandings and to bugs being introduced in future updates. More detailed and available documentation, including code comments, can help internal and external entities trying to collaborate on the project and can prevent developers from introducing additional security-critical bugs in the future updates.

2.3 Regular updates

New releases of polkadot-sdk may contain fixes for critical security issues. Since Peaq is a product that heavily relies on polkadot-sdk, updating to the latest version as soon as possible whenever a new release is available is advised.

2.4 Regular code review and continuous fuzz testing.

Regular code reviews are recommended to avoid introducing new logic or arithmetic bugs, while continuous fuzzing tests can identify potential vulnerabilities early in the development process. Ideally, Peaq Network should continuously fuzz their code on each commit made to the codebase. The substrate-runtime-fuzzer [2] (which uses Ziggy, a fuzzer management tool) can be a good starting point.

2.5 Launch a bug bounty program

Bug bounty programs encourage freelance researchers to continue the security testing of system components far into deployment. Introducing these incentives can increase code coverage in ways beyond traditional time-contained audits and can help identify critical edge-case bugs in live code. These programs may be rolled out later, dependent on the economic feasibility.

3 Motivation and scope

This report presents the results of the baseline security audit for Peaq's NFT Smart Contract, Precompiles and Tokenomics. It is important to note that the findings from previous engagements are not included in this document, as well as independent components such as DEX.

Peaq is a Layer-1 blockchain tailored for Decentralized Physical Infrastructure Networks (DePINs), focusing on real-world applications such as mobility and energy. It secures and streamlines identity verification for machines, vehicles, robots, and devices, ensuring seamless interaction across its ecosystem. It supports both ink! and EVM Smart Contracts, facilitating flexible development environments. Peaq's economy model incentivizes the contribution of machines and devices to its network.

Like other Substrate-based blockchain networks, the Peaq code is written in Rust, a memory safe programming language. Substrate-based chains utilize three main technologies: a WebAssembly (WASM) based runtime, decentralized communication via libp2p, and a block production engine. In addition to its technology stack, Peaq leverages decentralized identifiers (DIDs) and verifiable credentials for enhanced security and privacy in machine-to-machine interactions. These features enable machines to authenticate and verify each other's identity without relying on centralized authorities, ensuring better data integrity.

In a trustless, decentralized environment like a blockchain, security challenges are inherent. Therefore, ensuring availability and integrity is a priority for Peaq as it depends on its users to be incentivised to participate in the network. As such, a security review of the project should not only highlight the security issues uncovered during the audit process, but also bring additional insights from an attacker's perspective, which the Peaq team can then integrate into their own threat modeling and development process to enhance the security of the product.

Peaq has cultivated a decentralized ecosystem based on providing Ethereum application support within the broader Substrate community. This vision has been achieved by:

1. **Providing support for Ethereum-style RPC-calls** which allows existing Ethereum applications to be compatible with Substrate via Peaq.
2. **Mapping existing Substrate accounts to the 20-byte Ethereum address format** which allows users and Smart Contract applications to interact with accounts uniformly in both Ethereum and Peaq.
3. **Integrating runtime gas metering** to emulate the transaction fee mechanisms present in the Ethereum blockchain, while remaining compliant with the Substrate weight system. This allows Solidity Smart Contracts to exist on Peaq without requiring prior Substrate benchmarking.
4. **Implementing an extensive precompile feature set**, mapping core Substrate pallets to Solidity interfaces accessible via Ethereum style calls.

Peaq's runtime consists of multiple modules compiled into a WASM Binary Large Object (blob) that is stored on-chain. Nodes execute the runtime code either natively or will execute the on-chain WASM blob.

In the initial baseline assurance audit Security Research Labs collaborated with the Peaq development team to create an overview containing modules in scope and their audit priority. Following our baseline assurance, we gradually expanded our scope as new features became available and collaboratively outlined audit priority with the Peaq development team.

4 Methodology

This report details the results of our security audit on Peaq´s Smart Contracts, Tokenomics, and Precompiles between October to December 2024 with the aim of creating transparency in the following steps: security design coverage checks, reviewing runtime changes, and offering remediation support.

4.1 Security design coverage check.

Peaq feature designs were reviewed for coverage against relevant hacking scenarios. For each scenario, the following two aspects were investigated:

a. **Coverage**. Is each potential security vulnerability sufficiently covered?

b. **Underlying assumptions**. Which assumptions must hold true for the design to effectively reach the desired security goal?

4.2 Implementation check

Peaq features were tested for openings whereby any of the defined hacking scenarios could be executed. To effectively review Peaq's codebase and new features, we derived our code review strategy based on both the key areas of interest, and the priorities detailed by the Peaq development team; alongside our own internal threat models created for each feature review. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category.

Prioritizing by risk, the code was assessed for present protections against respective threats and attacks, as well as the vulnerabilities that make these attacks possible. For each threat, the audit process included the following steps:

1. **Identify the relevant parts of the codebase**, for example the relevant pallets and the runtime configuration.
2. **Identify viable strategies for the code review**. Manual code audits, and manual tests were performed where appropriate.
3. **Ensure the code doesn't contain any** vulnerabilities that could be used to execute the respective attacks, otherwise, ensured that sufficient protection measures against specific attacks were present.
4. **Immediately report any discovered vulnerability** to the development team along with suggestions around mitigations.

The steps were carried out through hybrid strategy utilizing a combination of manual code review and static testing to assess the security of the Peaq codebase. While static testing ensures baseline assurance, the focus of the security assurance is primarily on manual code review. The approach of feature reviews was to trace the intended functionality of modules in scope and to assess whether an attacker can bypass, misuse, or abuse these components, or trigger any unexpected behavior on the blockchain or the contracts.

4.3 Remediation support

The final step of this engagement consists of supporting Peaq's team with the remediation process of identified issues. For this purpose, each finding was documented and disclosed to the relevant members of the Peaq team responsible for each component of the audit with accompanying mitigation recommendations.

Our remediation recommendations are specifically tailored with an understanding of Peaq's business strategy and core success factors. Once the remediation is live, the fix is verified by our auditors to ensure that it mitigates the issue and does not introduce additional bugs.

Throughout our collaboration, findings were disclosed via our privately shared GitHub repository. We also engaged in asynchronous communication and status update meetings.

5 Findings summary

For our security audit from October to December 2024, Security Research Labs identified 4 high, 4 medium 5 low and 4 info security issues. These findings in

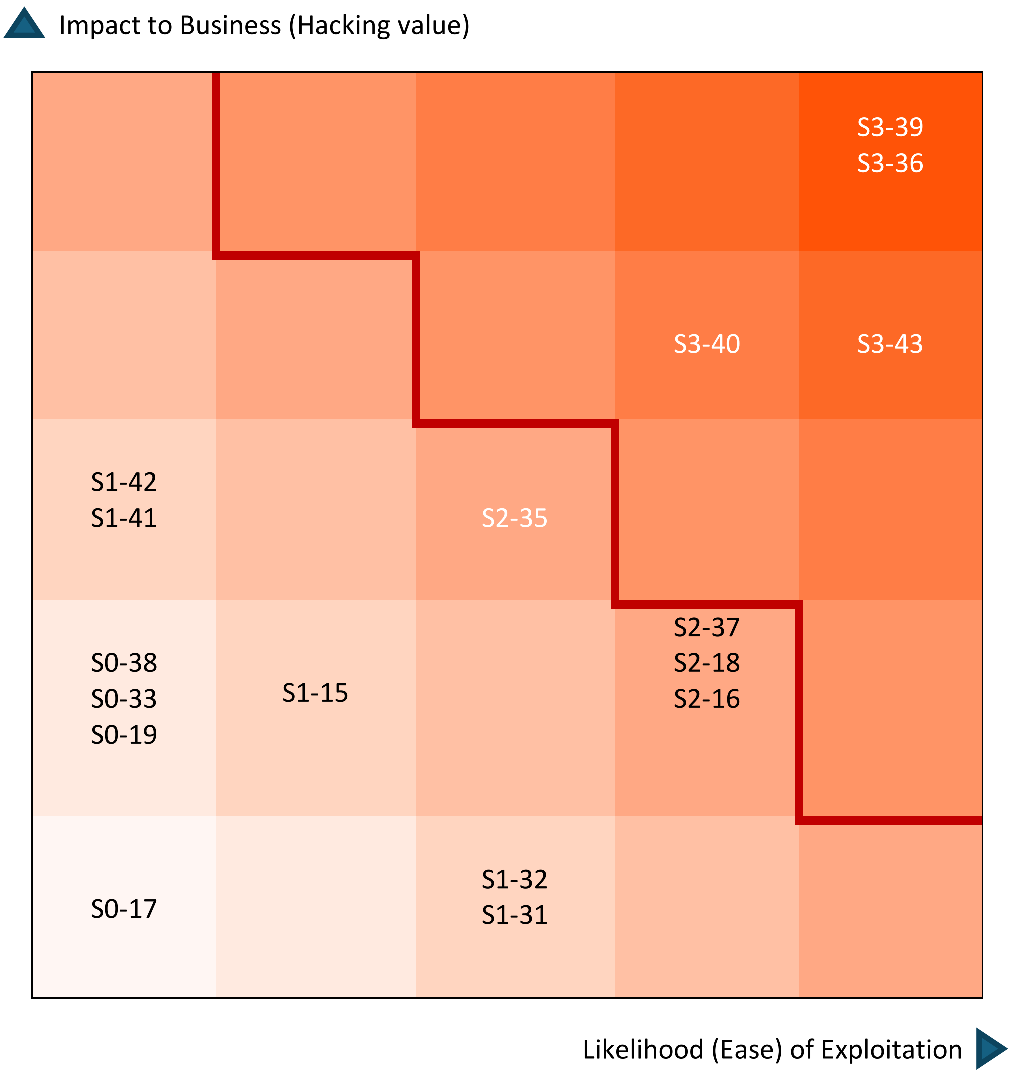
, are the culmination of various security testing processes implemented during our commitment to enhancing and preserving Peaq's security.

Critical	0	
High	4	●●●●
Medium	4	●●●●
Low	5	●●●●●
Informational	4	●●●●
Total Issues	**17**	

Please note that in our methodology, critical severity issues refer to high severity issues that could be exploited immediately by an attacker on already deployed infrastructure.

5.1 Risk profile

The chart below summarizes vulnerabilities according to business impact and likelihood of exploitation, increasing to the top right. The red margin separates the high-critical issues from medium/low/informational ones.



5.2 Issue summary

Tracking	Issue	Severity	Status
S3-43 [3]	`ExistentialDeposit` is configured to 0	High	Open
S3-40 [4]	Incorrect `on_finalize` weights may cause denial-of-service	High	Open
S3-39 [5]	Collator can drain delegator's rewards by manipulation commission rate	High	Open
S3-36 [6]	Missing trait-in-use checks in `_validateTraitOwnership` enables infinite minting	High	Mitigated [7]
S2-37 [8]	Single trait approval for NFT minting limits usability and efficiency	Medium	Mitigated [9]
S2-35 [10]	Trait token may be reused to satisfy multiple trait slots	Medium	Mitigated [9]
S2-18 [11]	Permissive `GasLimitStorageGrowthRatio` leads to excessive storage growth	Medium	Open
S2-16 [12]	Incorrect benchmarks for `pallet_evm`	Medium	Open
S1-42 [13]	Unsafe arithmetic in can halt collator payouts	Low	Open
S1-41 [14]	Lack of weight tracking in `note_author()` hook	Low	Open
S1-32 [15]	Missing sanity checks for `traitContract` address	Low	Mitigated [16]
S1-31 [17]	Single-step ownership transferal	Low	Open
S1-15 [18]	Incorrect topic selector for `RemoveAttribute` event submission	Low	Open
S0-38 [19]	Gas optimizations and logic efficiency	Info	Mitigated
S0-33 [20]	Missing events in `SolarSeekers` and `SolarSeekersTraits` contracts	Info	Mitigated
S0-19 [21]	Missing size checking could lead to high gas cost	Info	Open
S0-17 [22]	Incorrect fields names in reverted function backtraces	Info	Open

Table 2: Code review issue summary

6 Detailed findings

6.1 S3-43: ExistentialDeposit is configured to 0

Attack scenario	An attacker fills up storage with numerous unused accounts
Location	runtime/*
Attack impact	Increased overheads for node operators
Severity	High
Status	Open
Tracking	[3]

Background and Context
The `ExistentialDeposit` [23] is designed to ensure that inactive accounts with negligible balances are automatically removed to optimize on-chain storage. In the Peaq runtimes, this parameter has been set to zero, disabling the automatic cleanup of such accounts.

Problem Details
By setting the Existential Deposit to zero, accounts are never reaped, even if their balance drops to zero. This means that once an account has held any balance, the associated account data remains permanently stored.

Since Substrate's weight calculation for transactions does not consider the long-term cost of maintaining this persistent data an attacker could exploit this configuration by distributing minimal balances across a vast number of accounts, cheaply filling up blockchain storage with inactive or unnecessary data.

Transaction fees may reduce the likelihood of such an attack but do not eliminate the underlying risk because the costs associated with permanent storage are not sufficiently accounted for.

Risk
The absence of an existential deposit lowers the barrier to executing storage spamming attacks. This issue can lead to long-term storage bloat, degraded node performance and an increase in overheads associated with maintaining the network.

Recommendation
We recommended setting the existential deposit to a small, non-zero value to make it financially impractical for an attacker to create and maintain large numbers of dormant accounts.

6.2 S3-40: Incorrectly on_finalize weights might lead to denial-of-service attacks

Attack scenario	An attacker executes underweight computation
Location	pallets/parachains-staking/
Attack impact	Overweight blocks may be rejected resulting in service issues
Severity	High
Status	Open
Tracking	[4]

Background and Context

Peaq utilizes the `on_finalize` [24] hook to determine if any payouts need to be processed at the end of each block. Accurate weight calculation is essential to ensure that blocks do not exceed their resource limits, which requires the `on_initialize` [25] hook to account for the weight that will be consumed later in `on_finalize`.

Problem Details

The current implementation of `on_initialize` does not accurately simulate the weight that `on_finalize` will consume. Instead of dynamically estimating the required weight, it calls `on_initialize_no_action`, which returns a static weight based on a single database read. This simplification fails to reflect the actual operations performed during `on_finalize`, leading to an underestimation of resource usage.

Additionally, there is an oversight related to the `AtStake` storage map. When `CollatorBlocks` does not contain a valid author for a given round, items in `AtStake` that match a specific prefix are deleted. These deletions involve additional database writes, but the corresponding weight costs are not accounted for because the else branch of the logic does not update the read and write operations. This further contributes to inaccurate block weight reporting.

Risk

Inaccurate weight calculations in `on_initialize` can result in underestimating the total block weight. This miscalculation increases the risk of producing overweight blocks, potentially resulting in block rejection and service issues due to an inability to finalize.

Recommendation

We recommended implementing dynamic weight calculations at the beginning of each block to account for the expected operations in `on_finalize`. This includes simulating the weight of database reads and writes, particularly for payout processing and the deletion of items in `DelayedPayoutInfo` and `AtStake`. Each removal operation should be factored into the weight estimation to ensure that the total block weight remains accurate and reflective of actual resource usage.

6.3 S3-39: Collator can drain delegator's rewards by manipulation commission rate

Attack scenario	Malicious collators raise commission rates post-delegation
Location	pallets/parachains-staking/
Attack impact	Defrauded delegators forfeit significant financial rewards
Severity	High
Status	Open
Tracking	[5]

Background and Context
Collators can set and modify their commission rates via `setCommision()` [26], which determines the portion of staking rewards they retain. Delegators delegate their tokens to collators based on these commission rates, expecting to receive a fair share of the rewards generated.

Problem Details
A malicious collator can exploit the ability to change commission rates at any time to defraud delegators. The collator could initially set a very low commission rate to attract delegators, encouraging them to stake significant amounts of tokens. After securing these delegations, the collator could then raise their commission rate to 100%, effectively diverting all staking rewards to themselves. This manipulation leaves the delegators without any rewards, even if they realize the change and initiate the unstaking process. Due to the enforced `StakeDuration` [27] period (defaulting to one week), delegators remain exposed to the exploit for the entire unstaking period, with no way to recover the lost rewards.

Risk
Delegators are exposed to significant financial loss as their staking rewards can be entirely redirected to a malicious collator. The risk is significant because the attack can be easily executed without technical barriers and remains effective even after the delegator initiates unstaking. The potential impact extends beyond individual losses, as it can erode trust in the staking mechanism, discouraging participation and undermining the perceived security and fairness of the network.

Recommendation
We recommended implementing mechanisms to lock the commission rate for each delegation at the time of creation. Under this system, the commission rate agreed upon when a delegator stakes with a collator would remain fixed for that specific delegation until the delegator withdraws their stake.

Collators may still retain the ability to adjust their commission rates, but any changes would only apply to new delegations made after the rate adjustment. This approach preserves flexibility for collators while protecting delegators from retroactive changes.

6.4 S3-36: Missing trait-in-use checks in _validateTraitOwnership enables infinite minting

Attack scenario	A user reuses the same base trait to mint multiple derived tokens
Location	contracts/SolarSeekers.sol
Attack impact	Derived tokens will be devalued, and holders will experience financial loss
Severity	High
Status	Mitigated [7]
Tracking	[6]

Background and Context

In the NFT implementation, derived ERC-721 tokens can be minted using a set of base traits. The function `mintWithTraits()` [28] facilitates this process by attaching specific traits to newly minted tokens, creating a relationship between the base traits and the derived tokens.

Problem Details

The core issue arises from the absence of a mechanism to prevent the reuse of the same base traits across multiple derived tokens. While the `_validateTraitOwnership()` [29] function verifies trait ownership and existence during the minting process, it does not track whether a trait has already been used to mint another derived token. As a result, the same set of base traits can be reused indefinitely to mint an unlimited number of derived NFTs.

Risk

This exploit allows malicious actors to infinitely mint derived tokens using the same base traits. As the total supply of derived tokens grows unchecked, their scarcity diminishes leading to inevitable devaluation. This inflationary effect undermines the economic model of the NFT collection, eroding user trust and causing financial losses for holders.

Recommendation

We recommended implementing a global trait usage tracking system, for example a mapping the tracks usage, to prevent the reuse of traits in multiple derived tokens. During the minting or updating process, the system should verify the usage status of each trait. When traits are detached from a derived token, the usage status should be updated accordingly to allow for future reuse.

To further enhance security, it was recommended to restrict the transfer of traits that are currently assigned to active derived tokens. Implementing a query mechanism for checking trait usage status during token transfers will help enforce this restriction, ensuring that the relationship between traits and derived tokens remains consistent and secure.

6.5 S2-37: Single trait approval for NFT minting limits usability and efficiency

Attack scenario	Users have their existing approval overwritten due to misalignment with the approving party
Location	contracts/SolarSeekerTraits.sol
Attack impact	This may result in poor user sentiment and high overheads for approval management
Severity	Medium
Status	Mitigated [30]
Tracking	[8]

Background and Context
The SolarSeekerTraits NFT contract includes an approval mechanism for trait minting, where the contract owner authorizes users to mint specific traits. This system is essential for managing the controlled distribution of traits, which are prerequisites for minting the derived NFTs.

Problem Details
The current approval mechanism relies on the `allowMint()` [31] function, which maps each user's address to a single approved trait URI. This design limits users to having only one approved trait at a time. Since minting the final NFT requires at least four traits, users must undergo a repetitive process: receiving approval, minting the trait, then requesting approval again for the next trait. This cycle must be repeated for each required trait.

Moreover, if a new trait is approved for a user before they have minted the previously approved one, the new approval overwrites the existing one, potentially causing confusion and errors in the minting process.

Risk
The single-trait approval model increases the likelihood of minting errors due to overwritten approvals. It demands continuous monitoring and manual intervention from both the users and the contract owner, leading to a cumbersome user experience.

Additionally, the need for multiple transactions to approve and mint each trait results in higher gas costs and time inefficiencies. This not only affects usability but also poses a scalability concern if the system experiences increased user activity.

Recommendation
We recommended enhancing the approval system to support multiple concurrent trait approvals per user. This can be achieved by modifying the mapping structure to associate each user with an array of approved trait URIs, allowing several traits to be authorized simultaneously.

Alternatively, a nested mapping structure could be implemented, where each user address maps to multiple trait URIs, with a boolean flag indicating the approval status. These changes would reduce the need for repetitive approvals, streamline the minting process, and improve the overall user experience while minimizing the risk of errors and reducing transaction costs.

6.6 S2-35: Trait token may be reused to satisfy multiple trait slots

Attack scenario	Derived tokens may be minted with 1 base trait or equipped with the same trait multiple times
Location	contracts/*
Attack impact	Users may be misled on the usage of base traits for derived tokens
Severity	Medium
Status	Mitigated [9]
Tracking	[10]

Background and Context

The `SolarSeekers.sol` contract requires users to prove ownership of four base trait tokens from the `SolarSeekerTrait.sol` contract to mint an ERC721 derived NFT. In addition to these required traits, users can attach up to 16 optional traits, allowing further customization of the derived token.

Problem Details

While users may call `updateTraits()` [32] to attach optional traits after minting a derived NFT, there is no mechanism in place to prevent the same trait from being attached to different slots. This allows users to reuse a base trait multiple times in different slots.

Additionally, when minting the final NFT using `mintWithTraits()xw`, users may pass the same base trait multiple times in the necessary traits array, allowing them to mint with only a single valid trait, rather than four distinct ones.

Risk

This issue undermines the fundamental concept of non-fungibility within the ERC-721 tokens, as users can arbitrarily reuse traits to fill multiple slots, making it appear as though the derived token possesses more unique traits than it does. This misrepresentation could lead to confusion or false assumptions, such as the belief that the final derived NFT is associated with 20 unique traits or that the owner holds 20 distinct trait tokens.

Moreover, a single trait could be reused to mint a derived token, breaking the fundamental supply bounding of derived minting.

Recommendation

We recommended implementing checks during the trait minting and updating processes to ensure that a trait token is not reused in multiple slots. These checks should enforce the uniqueness of traits assigned to different slots, ensuring that each trait is only used once for the relevant slots. This would preserve the non-fungible nature of the derived NFTs and provide users with an accurate representation of the uniqueness and ownership of the traits.

6.7 S2-18: Permissive `GasLimitStorageGrowthRatio` leads to excessive storage growth

Attack scenario	An attacker submits multiple storage heavy transactions
Location	runtime/*
Attack impact	Node operators experience large storage overheads
Severity	Medium
Status	Open
Tracking	[11]

Background and Context

The `GasLimitStorageGrowthRatio` [33] governs the ratio of gas fees charged per byte of storage usage. Setting this value too low allows storage to be accessed disproportionately cheaply.

Problem Details

`GasLimitStorageGrowthRatio` is currently configured with a value of 1 which establishes a direct one-to-one correlation between gas consumption and storage growth, meaning for every unit of gas consumed, an equivalent unit of storage is allocated.

Risk

The key risk posed by this configuration is the potential for cheap storage bloating. This can severely affect the scalability of the blockchain, as the network could become burdened with excessive data that is costly to maintain. Without a mechanism to adjust gas pricing based on storage demands, attackers could exploit this by flooding the blockchain with storage-heavy transactions,

Recommendation

To mitigate this risk, we recommend adopting the approach implemented by Moonbeam. This method calculates the `GasLimitStorageGrowthRatio` using the formula: `BLOCK_GAS_LIMIT / BLOCK_STORAGE_LIMIT`. By applying a fixed block storage limit, the ratio ensures that storage growth remains in check relative to the gas consumed. The default value suggested for this ratio is 366, which has proven effective in managing storage scalability.

Additional insights and technical details on this approach can be found in MBIP-5 code [34], the associated pull request [35] and the following documentation [36]. These resources offer comprehensive guidance on the importance of setting an appropriate storage-growth ratio and demonstrate the benefits of this configuration for maintaining a scalable and secure blockchain infrastructure.

6.8 S2-16: Incorrect benchmarks for `pallet_evm`

Attack scenario	An attacker submits multiple overweight extrinsics
Location	runtime/*
Attack impact	Overweight blocks may be rejected resulting in service degradation
Severity	Medium
Status	Open
Tracking	[12]

Background and Context

The `pallet_evm` is responsible for enabling Ethereum Virtual Machine (EVM) compatibility within the runtime, allowing for the execution of Ethereum-based Smart Contracts. Benchmarking this pallet is essential to determine accurate weight calculations for transaction fees and resource usage.

Problem Details

The benchmarking process for `pallet_evm` in Peaq is conducted using the default Substrate benchmarks (`SubstrateWeight<Runtime>`) instead of benchmarks tailored to Peaq's specific runtime configurations **[37]**. Since runtime-specific factors can significantly influence the performance of extrinsics, relying on default benchmarks results in inaccurate weight calculations.

Risk

The primary risk lies in the potential for inaccurate weight assignments to extrinsics. This can manifest as either overweight or underweight extrinsics:

- Overweight extrinsics may lead to inefficient resource utilization, unnecessarily restricting throughput.
- Underweight extrinsics pose a more critical risk, potentially allowing transactions to consume more resources than accounted for, which can affect network stability and security.

These inaccuracies can impact transaction fees, block production efficiency, and overall network performance.

Recommendation

We recommended benchmarking `pallet_evm` using the actual runtime configurations to ensure accurate weight calculations. This involves integrating the specific pallet into the `define_benchmarks!` block. As a reference, we highlighted the Kusama runtime implementation [38] as a best practice example for achieving precise benchmarking aligned with the runtime's operational characteristics.

6.9 S1-42: Unsafe arithmetic can halt collator payouts

Attack scenario	The total stake exceeds `u128::MAX` thereby overflowing on aggregation
Location	pallets/parachains-staking/
Attack impact	Collators may unfairly receive zero rewards regardless of staked amount
Severity	Low
Status	Open
Tracking	[13]

Background and Context
The `get_collator_reward_per_session` [39] function is designed to calculate the total rewards owed to collators at the end of each block. It aggregates the stakes of all delegators to determine accurate reward payouts, which are then processed through the `payout_collator()` function.

Problem Details
The function utilizes `fold()` with `CurrencyBalance` initialized from `0u128` to sum the stakes of all delegators. If the total delegated stake exceeds the `u128::MAX` limit, an overflow occurs, causing the `delegator_sum` to reset to zero [40]. This incorrect zero value propagates through subsequent reward calculations, affecting the `delegator_nominator` and `delegator_percentage`, both of which also become zero [41].

As a result, the reward formula simplifies incorrectly:

$$percentage \times issue\_number + stake.commision \times (delegator\_percentage \times issue\_number)$$

This reduces to:

$$percentage \times issue\_number + stake.commision \times 0$$

Consequently, collators receive zero rewards despite legitimate staking activities. The overflow leads to a denial of service by effectively halting the payout mechanism, as the system treats the delegator sum as zero even when substantial stakes exist.

Risk
This issue poses a risk to the network's economic stability since collators may not receive the correct payouts, potentially undermining their incentives to secure the network. However, the likelihood of this issue occurring is extremely low due to the immense value of `u128::MAX`, which would require an unrealistically large sum of delegated stakes to trigger the overflow.

Recommendation
We recommended implementing safe arithmetic operations using `CheckedAdd` or `SaturatingAdd` to prevent overflows during stake aggregation. Additionally, we advised applying these best practices within the `Reward` struct to ensure robust handling of large stake sums, safeguarding the reward distribution process against potential arithmetic vulnerabilities.

6.10 S1-41: Lack of weight tracking in note_author() hook

Attack scenario	Collator selection does not keep track block weight consumed
Location	pallets/parachain_staking/
Attack impact	Unchecked weights may result in overweight blocks, resulting in chain stall
Severity	Low
Status	Open
Tracking	[14]

Background and Context

To track the selection of collectors, Peaq uses the `pallet_authorship.` The `note_author()` function keeps trace of the collators and changing the state does not keep track of the weight consumed.

Problem Details

The weight impact of those state changes tracked in `note_author()` [42] is not taken into consideration, and should be tracked using frame_system's `register_extra_weight_unchecked()` [43].

Risk

The risk is low since only 2 reads and 1 write are performed, this is however a bad practice that could introduce severe bugs if `note_author` ever needs to iterate over a long list of collators.

Recommendation

We suggest using `register_extra_weight_unchecked` to inform the runtime of this weight usage.

6.11 S1-32: Missing sanity checks for traitContract address

Attack scenario	The contract is incorrectly deployed with a null `traitContract` address
Location	contracts/SolarSeekers.sol
Attack impact	The contract must be redeployed resulting in unnecessary costs
Severity	Low
Status	Mitigated [16]
Tracking	[15]

Background and Context

The `SolarSeekers.sol` contract is designed to manage token minting with associated traits, relying on an external `traitContract` to handle trait-related logic. Proper initialization of this contract address is critical for ensuring seamless minting operations.

Problem Details

During deployment, the `traitContract` [44] address is initialized without any validation checks to confirm its correctness, such as ensuring it is not set to `address(0)`. If a misconfiguration occurs at deployment, there is no mechanism within the contract to update or correct this mistake post-deployment.

Risk

If the `traitContract` address is incorrectly set at deployment, the `mintWithTraits()` [28] function would fail to operate as intended, effectively breaking the minting process. Since the contract lacks an upgrade mechanism for this address, the only remedy would be a complete redeployment of the contract.

Recommendation

We recommended implementing validation checks during the contract's initialization phase to ensure that the `traitContract` address is set to a valid address. This would help prevent deployment misconfigurations and reduce the risk of operational failures tied to incorrect contract references.

6.12 S1-31: Single-step ownership transferal

Attack scenario	The current `owner` transfers ownership to a misconfigured `address`
Location	contracts/*
Attack impact	Ownership of the contract is completely lost, locking critical functionality
Severity	Low
Status	Open
Tracking	[17]

Background and Context

The `SolarSeekers.sol` and `SolarSeekersTraits.sol` Smart Contracts utilize OpenZeppelin's `Ownable` library to manage ownership and privileged access. Ownership transfers are handled through the `transferOwnership()` function, which assigns control of the contract to a new address. This mechanism is critical for maintaining administrative control over contract functions.

Problem Details

The current implementation of ownership transfers relies solely on the `transferOwnership()` [45] function, which performs basic input checks but does not verify whether the new owner address can properly interact with the contract.

This creates a gap where ownership could be unintentionally transferred to an incorrect address—such as a mistyped external address or a Smart Contract lacking the necessary functions to manage the system effectively. Since the transfer is immediate and final, there is no built-in safeguard to confirm that the new owner can fulfill administrative duties.

Risk

Transferring ownership to an incorrect address, such as one with typos, or to a Smart Contract that cannot handle ownership responsibilities, could result in a permanent loss of control over the contract.

This scenario could lead to an irreversible situation requiring contract redeployment, which may be costly and disruptive. The lack of a two-step verification process increases the likelihood of human error, expanding the attack surface beyond technical vulnerabilities to include administrative mistakes.

Recommendation

We recommended implementing a two-step ownership transfer process to mitigate the risks associated with immediate ownership changes. This can be achieved by integrating OpenZeppelin's `Ownable2Step` dependency [46], which introduces an additional confirmation step.

With this approach, ownership is not fully transferred until the new owner explicitly accepts the role, ensuring that the receiving address is both accurate and capable of interacting with the contract. This reduces the risk of accidental misconfigurations and enhances the overall security of privileged role management.

6.13 S1-15: Incorrect topic selector for `RemoveAttribute` event submission

Attack scenario	Events consistently emit incorrect function details
Location	precompiles/
Attack impact	Off-chain monitoring tools may be incompatible
Severity	Low
Status	Open
Tracking	[18]

Background and Context

When executing precompile functions on the Ethereum Virtual Machine (EVM), events are emitted to signal the occurrence of specific actions. These events are identified by unique selectors, which are derived from the `keccak256()` hash of the function signature. Accurate event selectors are crucial for external systems that rely on event listeners to track and respond to contract activity.

Problem Details

A typographical error exists in the selector for the `RemoveAttribute` function. The function signature is incorrectly hashed as `RemoveAttribte(address,bytes)` [47] due to a misspelling in the word "Attribute." This typo results in an incorrect event topic being generated.

As a result, any off-chain applications or monitoring tools that are configured to listen for the correct selector corresponding to `RemoveAttribute(address,bytes)` will fail to detect these events.

Risk

While this issue does not directly impact the core functionality of the contract, it can cause significant issues for systems that rely on accurate event detection, such as analytics tools or monitoring systems that automate processes based on event emissions. This discrepancy can cause event listeners to miss critical precompiled function executions, leading to gaps in event-driven workflows or data inaccuracies in systems dependent on these logs.

Recommendation

We recommended correcting the typo in the event selector to ensure consistency with the intended function signature when hashed. This adjustment will align the emitted event topic with the expectations of external systems, ensuring that all relevant events are accurately captured and processed.

6.14 S0-38: Gas optimizations and logic efficiency

Attack scenario	Contract execution incurs unnecessa**ry** overheads for users and operators
Location	contracts/*
Attack impact	Transactions fess may be excessive
Severity	Info
Status	Mitigated
Tracking	[19]

Background and Context
We determined that several observations related to optimizing gas usage in the Smart Contract code would be included in the guidance offered by Security Research Labs. These improvements yield benefits in reducing transaction fees and minimizing execution overheads.

Observations and analysis

- **Use custom reversion instead of require statements**. Require statements emit failure strings, which not only increase gas usage due to the string length but also reduce uniformity in error handling across the contract.
- **Preform pre-fix (++i) instead of post-fix (i++) loop iterator increments**. When using post-fix increments (`i++`), Solidity creates a temporary variable to hold the un-incremented value of `i`. Pre-fix increments (`++i`) avoid this temporary variable, providing a small gas saving.
- **Cache storage items to memory before performing complex processes**. Performing multiple reads and writes to storage can be costly, as each read/write incurs gas costs. A better approach is to read data from storage into memory before performing iterative operations, as memory is significantly cheaper for repeat access.
- **Perform safe arithmetic in unchecked blocks**. Solidity 0.8.0 introduces automatic overflow checks for arithmetic operations, but these checks can increase gas costs. In cases where overflow is not a concern, the `unchecked` block can be used to disable these checks.
- **Use != instead of > when comparing unsigned integers against zero**. When comparing unsigned integers to zero, the `!=` operator is more gas-efficient than `>`. This comparison does not change the logic, as unsigned integers are always non-negative, but reduces the computational cost.
- **Use external instead of public for functions not called internally**. Functions marked as `public` are more expensive because they allow both internal and external calls. When functions are only called externally, marking them as `external` reduces gas costs, as external functions do not copy arguments to `memory`.

RecommendationWe made the following recommendations based on the previous observations regarding logic efficiency

- stead of using `require` with a lengthy failure string, create custom `error` types and utilizing reversion with reusable error codes to save gas by reducing the overhead associated with string handling.
- Change all post-fix increments to pre-fix increments (`++i`) when looping to optimize gas usage.
- Cache storage items in `memory` prior to performing iterations or complex operations. For example, storing the `array.length` in `memory` reduces the need for multiple reads.
- Use `unchecked` blocks for arithmetic operations that are safe, for example fixed arithmetic with `constant` values that cannot overflow.
- Use `!= 0` instead of `> 0` when comparing unsigned integers to save gas.
- Change functions that are not called internally from `public` to `external` to optimize gas costs.

6.15 S0-33: Missing events in SolarSeekers and SolarSeekersTraits contracts

Attack scenario	Inadequate event logging limits monitoring capabilities
Location	contracts/SolarSeekers.sol
Attack impact	No impact, limits monitoring capability and tracing of contract executions
Severity	Info
Status	Mitigated [48] [49]
Tracking	[20]

Background and Context
Event emissions are important features in Smart Contract applications, as often shipping data off-chain for monitoring is otherwise impossible. Throughout the codebase, there are various instances of critical processes not emitting appropriate data off-chain on state-change.

Problem Details
The following Smart Contracts for the following specific cases do not emit event on the execution:

- `updateTraits()` in SolarSeekers.sol where attached traits are modified should emit `msg.sender`, `tokenId` and the new attached traits.
- `allowMint()` in SolarSeekersTraits.sol where a user is whitelisted a token allocation should emit the `receiver` and `uri`.

Risk
Missing events is a non-critical issue that does not inherently introduce security risks, although it can impact the quality of off-chain monitoring efforts. This may contribute to inaccurate system state understanding and make responding to certain scenarios more difficult.

Recommendation
Implement event emissions for functionality surrounding minting and approving mints on-top of the existing support provided by Open Zeppelin's libraries.

6.16 S0-19: Missing size checkings could lead to unnecessary gas cost

Attack scenario	**Missing bound checks in precompiles leads to unnecessary gas consumption**
Location	precompiles/peaq-rbac
Attack impact	Without gas optimization, an attacker can execute transactions cheaply
Severity	Info
Status	Open
Tracking	[21]

Background and Context

The access control pallet RBAC deployed in the Peaq Network enable user to add permissions, create and define roles. In the `add_permission` precompile, there was no proper input sanitisation performed before dispatching the call. This could result in excessive gas consumption than accounted for and possible transaction reversal.

Problem Details

The `add_permission` precompile within peaq-rbac pallet allows users to add a permission using a name of type `BoundedBytes<GetBytesLimit>`.However, the size of this parameter is not validated beforehand and is directly dispatched in the corresponding substrate extrinsic. If the parameter size exceeds `MAX_NAME_SIZE` (64), the extrinsic will be reverted, incurring additional gas charges that could have been avoided. Note that `BoundedBytes<GetBytesLimit>` is constrained to 2^{16} (65,536). Consequently, users may enter a name exceeding `MAX_NAME_SIZE` resulting in a revert during `try_dispatch`.

The following **functions are also affected:**

- `update_group`
- `update_permission`
- `add_group`

Risk

While there are no significant risks associated with this issue, unnecessary gas costs can be easily avoided, potentially leading to cheaper transactions for users.

Recommendation

We recommend bounding the vector to `MAX_NAME_SIZE` during conversion to `BoundedVec` and reverting if the limit is exceeded.

6.17 S0-17: Incorrect fields names in reverted function backtraces

Attack scenario	EVM logs with incorrect parameter names when precompiles call fail
Location	Precompiles/erc-20, precompiles/vesting, precompiles/asset-factory
Attack impact	None, incorrect logging and improper parameter name sanitization
Severity	Info
Status	Open
Tracking	[22]

Background and Context

When a precompile, call fails, the logging of call trace reflects incorrect parameters names. This will make the troubleshooting and long-term maintenance of the codebase cumbersome.

Problem Details

When a precompile is triggered, certain checks are performed to promptly revert the call in case of failure. For this purpose, a type called `MayRevert` is available, which implements `InjectBacktrace`. This trait is used to inject a message into `MayRevert`, which is then submitted as an event to the `pallet_evm` using a `LogsBuilder`.

However, some backtraces are incorrect and use misleading parameter names, such as:

- `amount` instead of `value`
- `locked_amount` instead of `amount`
- `per_block_amount` instead of `amount`
- `id` instead of `asset_id`

Risk

There are no risks associated with this issue; however, it might confuse users trying to understand why their call was reverted.

Recommendation

We recommend using the correct parameter names instead of generic placeholder names.

Bibliography

[1] [Online]. Available: https://docs.peaq.network/docs/learn/tokenomics/#inflation-and-transaction-fee-distribution.

[2] [Online]. Available: https://github.com/srlabs/substrate-runtime-fuzzer/tree/main.

[3] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/31.

[4] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/28.

[5] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/27.

[6] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/23.

[7] [Online]. Available: https://github.com/Milbo-GmbH/peaq-portal-frontend/tree/mint-trait/hardhat/contracts.

[8] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/25.

[9] [Online]. Available: https://github.com/Milbo-GmbH/peaq-portal-frontend/tree/mint-trait/hardhat/contracts.

[10] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/24.

[11] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/15.

[12] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/17.

[13] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/30.

[14] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/29.

[15] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/21.

[16] [Online]. Available: https://github.com/Milbo-GmbH/peaq-portal-frontend/blob/ac5972e0ad88d42fff60eb5186f4ce691c8c90a5/hardhat/contracts/SolarSeekers.sol#L35.

[17] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/20.

[18] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/18.

[19] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/26.

[20] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/22.

[21] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/19.

[22] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/16.

[23] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/bdb581ffe32965d8bc3a4b953fa12935eaa2df63/runtime/peaq/src/lib.rs#L454.

[24] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L535.

[25] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L527.

[26] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L1965.

[27] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/bdb581ffe32965d8bc3a4b953fa12935eaa2df63/runtime/peaq/src/lib.rs#L827.

[28] [Online]. Available: https://github.com/peaqnetwork/peaq-portal-frontend/blob/141b2500c91fc5e30ed73d1ecacbcfa309ad6bb6/hardhat/contracts/SolarSeekers.sol#L27.

[29] [Online]. Available: https://github.com/peaqnetwork/peaq-portal-frontend/blob/141b2500c91fc5e30ed73d1ecacbcfa309ad6bb6/hardhat/contracts/SolarSeekers.sol#L78-L79.

[30] [Online]. Available: https://github.com/Milbo-GmbH/peaq-portal-frontend/tree/mint-trait/hardhat/contracts.

[31] [Online]. Available: https://github.com/peaqnetwork/peaq-portal-frontend/blob/141b2500c91fc5e30ed73d1ecacbcfa309ad6bb6/hardhat/contracts/SolarSeekerTraits.sol#L26.

[32] [Online]. Available: https://github.com/peaqnetwork/peaq-portal-frontend/blob/141b2500c91fc5e30ed73d1ecacbcfa309ad6bb6/hardhat/contracts/SolarSeekers.sol#L37-L38.

[33] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/c92613c0fe17d81812fce4f85e1d648571ecfd55/runtime/peaq/src/lib.rs#L703.

[34] [Online]. Available: https://github.com/moonbeam-foundation/moonbeam/blob/master/MBIPS/MBIP-5.md.

[35] [Online]. Available: https://github.com/moonbeam-foundation/moonbeam/pull/2452.

[36] [Online]. Available: https://docs.moonbeam.network/learn/core-concepts/tx-fees/#overview-of-mbip-5.

[37] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/c92613c0fe17d81812fce4f85e1d648571ecfd55/runtime/peaq/src/lib.rs#L749.

[38] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/5bf21d73c0456eb7c5910aafa78445f93a61bdc9/relay/kusama/src/lib.rs#L1983-L2016.

[39] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L2705.

[40] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L2713.

[41] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L2743-L2744.

[42] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/756f984baab824d11e7c9815bb891b7a0f2cbd82/pallets/parachain-staking/src/lib.rs#L2951.

[43] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/da2dd9b7737cb7c0dc9dc3dc74b384c719ea3306/cumulus/pallets/collator-selection/src/lib.rs#L943-L946.

[44] [Online]. Available: https://github.com/peaqnetwork/peaq-portal-frontend/blob/141b2500c91fc5e30ed73d1ecacbcfa309ad6bb6/hardhat/contracts/SolarSeekers.sol#L23.

[45] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/be2b016679409cfbd23dba05c9100f6f4e0c6977/contracts/access/Ownable.sol#L84.

[46] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/be2b016679409cfbd23dba05c9100f6f4e0c6977/contracts/access/Ownable2Step.sol#L43.

[47] [Online]. Available: https://github.com/peaqnetwork/peaq-portal-frontend/blob/141b2500c91fc5e30ed73d1ecacbcfa309ad6bb6/app/util/ABI/DID.json#L55.

[48] [Online]. Available: https://github.com/Milbo-GmbH/peaq-portal-frontend/blob/ac5972e0ad88d42fff60eb5186f4ce691c8c90a5/hardhat/contracts/SolarSeekerTraits.sol#L48.

[49] [Online]. Available: https://github.com/Milbo-GmbH/peaq-portal-frontend/blob/ac5972e0ad88d42fff60eb5186f4ce691c8c90a5/hardhat/contracts/SolarSeekers.sol#L81.