Peaq Baseline Security Assurance

Threat model and hacking assessment report

V1.0.0, September 3, 2024

Kevin Valerio	kevin@srlabs.de
Louis Merlin	louis@srlabs.de
Aarnav Bos	aarnav@srlabs.de
Regina Biro	regina@srlabs.de

Abstract. This work describes the result of the thorough and independent security assurance audit of the Peaq blockchain performed by Security Research Labs. Security Research Labs is a consulting firm that has been providing specialized audit services for Substrate-based blockchains since 2019, including in the Polkadot ecosystem.

During this study, Peaq provided access to relevant documentation to support the research effort. The code of Peaq was verified to assure that the business logic of the product is resilient to hacking and abuse.

The research team identified several issues ranging from high severity to info, many of which concerned runtime extrinsics.

Security Research Labs recommends implementing rigorous validation logic in the Machine Owner Rewards mechanism, re-testing benchmarks and introducing penalization to deter fraudulent activity.

Content

1 Disclaimer 3
2 Motivation and scope 4
2.1 Baseline assurance 4
3 Methodology 5
4 Threat modeling and attacks 6
5 Baseline Assurance 10
5.1 Findings summary 10
5.2 Detailed findings 11
5.2.1 Incorrect authorization allows overriding DID attributes 11
5.2.2 Undervalued weight benchmarking could provoke block timeout 11
5.2.3 No *FeeManager* is configured for XCM messages 11
5.2.4 Missing service charge could enable abuse of service requests 12
5.2.5 Unverified *service_delivered* extrinsic execution 12
5.2.6 No XCM delivery fees configured for sibling parachain messages 12
5.2.7 Reward dispatching does not account for machine uptime 13
5.2.8 Insufficient checks in extrinsics could lead to financial exploitation 13
5.2.9 Incorrect staking requirements could compromise validation security 14
5.2.10 Extrinsic with missing storage deposits could clutter the storage. 14
5.2.11 Usage of old Substrate dependencies 15
5.2.12 Incorrect benchmarks for external pallets 15
5.2.13 Import of an insecure randomness algorithm 16
6 Evolution suggestions 16
6.1 Core improvement suggestions to improve security posture 16
6.2 Further recommended best practices 17
7 Bibliography 18

1 Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the agreed-on timeframe and scope as detailed in Chapter 2. Please note that this report does not guarantee that all existing security vulnerabilities were discovered in the codebase exhaustively and that following all evolution suggestions described in Chapter 6 may not ensure all future code to be bug free.

2 Motivation and scope

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.

2.1 Baseline assurance

In this current engagement, the audit team focused on the Peaq runtimes and pallet code. Security Research Labs collaborated with the Peaq team to create an overview containing the runtime modules in scope and their audit priority [1]. The in-scope components' source code repositories were forked to provide a frozen version for the audit. The assigned priorities and locations of the in-scope repositories are reflected in

Table 1. During the audit, Security Research Labs used a CIA-triad threat model to guide efforts on exploring potential security flaws and realistic attack scenarios.

During the assessment of the code, security critical parts of the code were identified and security issues in these components were communicated to the Peaq development team in the form of GitHub issues in a private repository [2].

Repository	Priority	Component(s)	Reference
peaq-pallet-did-audit-srl-2024	High	Pallet DID	[3]
peaq-pallet-rbac-audit-srl-2024	High	Pallet RBAC	[4]
peaq-pallet-mor-audit-srl-2024	High	Pallet MOR	[5]
peaq-storage-pallet-audit-srl-2024	High	Pallet Storage	[6]
peaq-network-node-audit-srl-2024	High	Network Node (runtimes) Parachain Staking Block Reward pallet Staking Coefficient Reward	[7]
peaq-pallet-transaction-audit-srl-2024	Medium	Pallet Transaction	[8]
peaq-network-node-audit-srl-2024	Low	Address Unification Pallet Staking Fixed Percentage Reward XC Asset Config	[7]

Table 1. In-scope Peaq components with audit priority

3 Methodology

This report details the baseline security assurance results for the Peaq network with the aim of creating transparency in three steps: threat modeling, implementation baseline check and finally remediation support:

Threat Modeling. The threat model is considered in terms of *hacking incentives*, i.e., the motivations to achieve the goals of breaching the integrity, confidentiality, or availability of Peaq network node. For each hacking incentive, hacking *scenarios* were postulated, by which these goals could be achieved. The threat model provides guidance for the design, implementation, and security testing of Peaq.

Implementation baseline check. As a second step, the Peaq implementation was tested for openings whereby any of the defined hacking scenarios could be executed.

To effectively review the Peaq codebase, Security Research Labs derived the code review strategy based on the threat model that was established as the first step. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category, as outlined in Chapter 4.

Prioritizing by risk, the codebase was assessed for present protections against the respective threats and attacks as well as the vulnerabilities that make these attacks possible. For each threat, the auditors:

1. Identified the relevant parts of the codebase, for example the relevant crates and the runtime configuration.

2. Identified viable strategies for the code review. Manual code review, fuzz testing, and tests via static analysis tools were performed where appropriate.

3. Ensured the code did not contain any vulnerabilities that could be used to execute the respective attacks, or otherwise ensured that sufficient protection measures against specific attacks were present.

4. Immediately reported any vulnerability that was discovered to the development team along with suggestions around mitigations.

Security Research Labs carried out a hybrid strategy utilizing a combination of code review and dynamic tests (e.g., fuzz testing) to assess the security of the Peaq codebase.

While fuzz testing and dynamic tests establish a baseline assurance, the focus of this audit was a manual code review of the Peaq codebase to identify logic bugs, design flaws, misconfigurations, and best practice deviations. The approach of the review was to trace the intended functionality of the runtime modules in scope and to assess whether an attacker can bypass/misuse/abuse these components or trigger unexpected behavior on the blockchain due to logic bugs or missing checks. Since the Peaq codebase is entirely open source, it is realistic that a malicious actor would analyze the source code while preparing an attack.

Fuzz testing is a technique to identify issues in code that handles untrusted input, which in Peaq's case is extrinsics in the main Peaq runtime. Fuzz testing works by taking some valid input for a method under test, applying a semi-random mutation to it, and then invoking the method under test again with this semi-valid input. Through repeating this process, fuzz testing can unearth inputs that would cause a crash or other undefined behavior (e.g., integer overflows) in the method under test. The fuzz testing methods written for this assessment utilized the test runtime genesis configuration as well as mocked externalities to execute the fuzz test effectively against the extrinsics in scope.

Remediation support. The final step is supporting Peaq with the remediation process of the identified issues. Each finding was documented and published with mitigation recommendations. Once the mitigation solution is implemented, the fix is verified by the auditors to ensure that it mitigates the issue and does not introduce other bugs.

4 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in Peaq's blockchain system. Familiarity with these risk areas can provide guidance for the design of the implementation stack, the actual implementation of the stack, as well as the security testing. This section introduces how risk is defined and provides an overview of the identified threat scenarios. The *Hacking Value*, categorized into *low*, *medium*, and *high*, considers the incentive of an attacker, as well as the effort required by an attacker to successfully execute the attack. The hacking value is calculated as:

$$Hacking\ Value = \frac{Incentive}{Effort}$$

While *incentive* describes what an attacker might gain from performing an attack successfully, *effort* estimates the complexity of this same attack. The degrees of incentive and effort are defined as follows:

Incentive:

- Low: Attacks offer the hacker little to no gain from executing the threat.
- Medium: Attacks offer the hacker considerable gains from executing the threat.
- High: Attacks offer the hacker high gains by executing this threat.

Effort:

- Low: Attacks are easy to execute. They require neither elaborate technical knowledge nor considerable amounts of resources.
- Medium: Attacks are moderately difficult to execute. They might require bypassing countermeasures, the use of expensive resources or a considerable amount of technical knowledge.
- High: Attacks are difficult to execute. The attacks might require in-depth technical knowledge, vast amounts of expensive resources, bypassing countermeasures, or any combination of these factors.

Incentive and Effort are divided according to Table 2.

Hacking Value	Low incentive	Medium Incentive	High Incentive
High effort	Low	Medium	Medium
Medium effort	Medium	Medium	High
Low effort	Medium	High	High

Table 2. Hacking value measurement scale.

Hacking scenarios are classified by the risk they pose to the system. The risk level, also categorized into low, medium, and high, considers the hacking value, as well as the damage that could result from successful exploitation. The risk of a threat scenario is calculated by the following formula:

$$Risk = Damage \times Hacking\ Value = \frac{Damage \times Incentive}{Effort}$$

Damage describes the negative impact that a given attack, performed successfully, would have on the victim. The degrees of damage are defined as follows:

Damage:

- Low: Risk scenarios would cause negligible damage to the Peaq blockchain.
- Medium: Risk scenarios pose a considerable threat to Peaq functionality as a blockchain.
- High: Risk scenarios pose an existential threat to Peaq's functionality.

Damage and Hacking Value are divided according to Table 3.

Risk	Low hacking value	Medium hacking	High hacking
Low damage	Low	Medium	Medium
Medium damage	Medium	Medium	High
High damage	Medium	High	High

Table 3. Risk measurement scale

After applying the framework to the Peaq system, different threat scenarios according to the CIA triad were identified.

The CIA triad describes three security promises that can be violated by a hacking attack, namely confidentiality, integrity, availability.

Confidentiality:

Confidentiality threat scenarios concern sensitive information regarding the blockchain network and its users. A threat scenario could include, for example, the leak of user's private information via his DID document.

Integrity:

Integrity threat scenarios pose significant risks to the Peaq network. The potential for financial gain is a primary motivator behind such attacks, including scenarios where an attacker might engage in harmful activities as a collator, unjustly claim a disproportionate share of machine owner rewards, or manipulate the fee refunding system to their advantage for example. Such actions, whether for direct financial gain or to undermine network integrity, pose challenges to maintaining Peaq's ecosystem's security.

Availability:

Availability threat scenarios refer to compromising the availability of the network to process normal transactions. Important threat scenarios regarding availability for blockchain systems include Denial of Service (DoS) attacks on participating nodes, stalling the transaction queue, and spamming.

Table 4 provides a high-level overview of the hacking risks concerning Peaq with identified example threat scenarios and attacks, as well as their respective hacking value and effort. The complete list of threat scenarios identified along with attacks that enable them are described in the threat model deliverable [1]. This list can serve as a starting point for the Peaq developers to guide their security outlook for future feature implementations. By thinking in terms of threat scenarios and attacks during code review or feature ideation, many issues can be caught or even avoided altogether.

For Peaq, the auditors attributed the most hacking value to the integrity class of threats. Undermining the integrity of the Peaq chain could lead to severe financial abuses and could have devastating consequences for the entities being impacted by such an attack.

Promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confiden-tiality	High	- Leak user's private information via his DID document - Impersonate an external machine or service provider	High	- Abuse a discrepancy between the peaq DID implementation and the W3C norm - Create a DID using another machine address during the Machine-Origin Authentication process
Integrity	High	- Conduct damaging behavior inside the network as a collator - Receive an illicit proportion of machine owner rewards - Manipulate the fee refunding system	Medium	- Quickly join and leave the set of delegators or collators for financial gain - Exploit distribution algorithm vulnerabilities to illicitly redirect and claim rewards intended for others - Send the refund extrinsic without deposit any funds beforehand
Availa-bility	Medium	- Harm the chain functionality by cluttering its storage - Censor certain transactions - Stall block production	Low	- Cheaply fill up blockchain storage - Censor transactions as a collator - Block transactions through heavy-weight extrinsic

Table 4. Risk overview. The threats for Peaq's blockchain were classified using the CIA security triad model, mapping threats to the areas: (1) Confidentiality, (2) Integrity, and (3) Availability.

5 Baseline Assurance

5.1 Findings summary

During the analysis of the Peaq code, Security Research Labs identified 13 issues (9 highs, 3 mediums, 1 info) which are summarized in Table 5.

Each finding to the Peaq project described here was shared with Peaq developers in a dedicated private GitHub repository as an issue.

Issue	Severity	References	Status
Incorrect authorization allows overriding DID attributes	High	[9]	Closed*
Undervalued weight benchmarking could provoke block timeout	High	[10]	Closed*
No *FeeManager* is configured for XCM messages	High	[11]	Closed*
Lack of service charge could allow an attacker to abuse service requests	High	[12]	Closed*
Unverified *service_delivered* extrinsic execution	High	[13]	Closed*
No XCM delivery fees configured for sibling parachain messages	High	[14]	Closed*
Reward dispatching does not account for machine uptime	High	[15]	Closed*
Insufficient checks in extrinsics could lead to financial exploitation	High	[16]	Closed*
Incorrect staking requirements could compromise validation security	High	[17]	Closed*
Extrinsics with missing storage deposits could clutter the blockchain storage	Medium	[18]	Closed*
Usage of old Substrate dependencies	Medium	[19]	Closed*
Incorrect benchmarks for external pallets	Medium	[20]	Closed*
Import of an insecure randomness algorithm	Info	[21]	Closed*

Table 5 Code audit issue summary

**As of September 3rd, 2024, the Peaq Team has reported that the issues have been addressed and resolved. However, these fixes have not yet undergone independent verification by Security Research Labs, and therefore the resolution has not been confirmed.*

5.2 Detailed findings

5.2.1 Incorrect authorization allows overriding DID attributes

Attack scenario	**Manipulate a DID to perform DID takeover**
Location	peaq-pallet-did
Tracking	[9]
Attack impact	Attackers can modify or remove DID attributes of any user
Severity	High
Status	Closed*

When creating a DID attribute for a DID account, a hash combining the sender of the extrinsic and the DID account is added to the *OwnerStore* map, where the key is the hash, and the value is the DID account. The DID attribute may be updated or removed by the *remove_attribute* and *update_attribute* extrinsics, which uses the *is_owner* function to perform a permission check on the attribute before proceeding.

The function *is_owner* only checks if the sender is part of *OwnerStore* and does not check if they created the attribute they wished to modify. Thus, if a user were to create a DID attribute for the DID account, they would be able to remove or update any attribute belonging to the DID account as they would be present in the *OwnerStore* map.

We recommend including the name of the DID attribute as part of the hash stored in *OwnerStore* and verifying this in the *is_owner* function.

5.2.2 Undervalued weight benchmarking could provoke block timeout

Attack scenario	**Send incorrectly benchmarked extrinsics to spam the network**
Location	peaq-pallet-did, peaq-pallet-rbac, peaq-storage-pallet
Tracking	[10]
Attack impact	Attackers can stall block production.
Severity	High
Status	Closed*

In Substrate-based blockchains, weights are used to calculate appropriate fees and estimate execution time. These calculations ensure transactions are executed properly, so block production guarantees can be met. Extrinsics with underestimated weights may allow an attacker to create blocks that are too large, leading to block production timeouts.

Our audit identified five extrinsics where the benchmarks do not consider the maximum sizes of their parameters. This is not a comprehensive list, and we recommend a thorough analysis of all pallets to determine if the benchmarks accurately reflect worst-case scenarios.

5.2.3 No *FeeManager* is configured for XCM messages

Attack scenario	**Abuse XCM misconfiguration**
Location	agung, krest, peaq, peaq-dev runtimes
Tracking	[11]
Attack impact	Attackers can bypass paying fees and spam the blockchain

Severity	High
Status	Closed*

The runtimes Agung, Krest, Peaq, and Peaq-dev include the *xcm_executor* pallet. These runtimes configure the *xcm_executor's FeeManager* to be (), which waives all XCM call fees and makes the chain vulnerable to spam from sibling parachains.

To mitigate this issue, we recommend using *XcmFeeManagerFromComponents* for the *FeeManager* type. Rococo's XCM configuration can be used as a reference.

5.2.4 Missing service charge could enable abuse of service requests

Attack scenario	**Request a machine service for free, bypassing the deposit step**
Location	peaq-pallet-transaction
Tracking	[12]
Attack impact	Attackers can receive services for free
Severity	High
Status	Closed*

Peaq-pallet-transaction includes a *service_requested* extrinsic, where the caller can request a service from a specified provider and deposit tokens that will be used later to pay the provider. However, due to incomplete implementation of the extrinsic, an attacker could request services without depositing any tokens.

We recommend implementing a proper token transfer from the consumer to the provider or using *set_lock* to lock the amount and perform the transfer later.

5.2.5 Unverified *service_delivered* extrinsic execution

Attack scenario	**Manipulate the fee refunding system**
Location	peaq-pallet-transaction
Tracking	[13]
Attack impact	Attackers can gain illicit refunds
Severity	High
Status	Closed*

The *service_delivered* extrinsic, exposed by *peaq-pallet-transaction*, allows a provider to acknowledge the delivery of a service to its consumer. However, this extrinsic does not perform checks to ensure that the sender is the actual executor of the service. An attacker could provide incorrect parameters, leading to various security issues such as provider impersonation and token refund abuse.

We recommend implementing appropriate validation checks to confirm that the caller of the extrinsic is indeed the provider of the service

5.2.6 No XCM delivery fees configured for sibling parachain messages

Attack scenario	**Abuse XCM misconfiguration**
Location	agung, krest, peaq, peaq-dev runtimes
Tracking	[14]
Attack impact	Attackers can avoid paying fees and spam the blockchain
Severity	High

Status	Closed*

In the runtimes Agung, Krest, Peaq, and Peaq_dev, the pallet cumulus_pallet_xcmp_queue's *PriceForSiblingDelivery* is configured to be (). This configuration results in no fees being charged for delivering XCM messages across parachains.

Attackers may exploit this by sending spam messages across chains without incurring any fees. Excessive messages could lead to XCM queue size exhaustion due to excessive storage usage until messages are delivered. This could also result in delays in message delivery for other users.

5.2.7 Reward dispatching does not account for machine uptime

Attack scenario	**Manipulate online time-tracking system to fraudulently claim rewards**
Location	peaq-pallet-mor
Tracking	[15]
Attack impact	Attackers can falsify the online status or productivity of devices
Severity	High
Status	Closed*

Peaq's peaq-pallet-mor exposes the *get_online_rewards* extrinsic, which allows machine owners to claim rewards for their machines' contributions to the network. However, the extrinsic does not verify the uptime of the machines, potentially enabling attackers to deplete the rewards pot, causing Denial of Service and severe financial exploitation.

We recommend implementing a verification mechanism within the *get_online_rewards* extrinsic to assess the actual uptime of machines before distributing rewards. This could be achieved by integrating an uptime verification system that periodically checks the uptime status of registered machines; using secure timestamps or external oracles could provide evidence of machine uptime. Additionally, introducing penalties for dishonest claims could further protect the integrity of the rewards system.

5.2.8 Insufficient checks in extrinsics could lead to financial exploitation

Attack scenario	**Cause system to mint tokens to his own account**
Location	peaq-pallet-mor
Tracking	[16]
Attack impact	Attackers can abuse token minting mechanism to disrupt the blockchain economy
Severity	High
Status	Closed*

The *pay_machine_usage* extrinsic is designed to facilitate payment for the use of a physical machine within the Peaq network. This implementation assumes that the origin is requesting a service from a machine and that, since users do not yet have tokens, those tokens will be minted and transferred immediately thereafter. Similarly, the *get_registration_reward* function, which aims to register and distribute rewards to a specific machine, is also affected by this issue.

No checks are performed to verify that the machine is genuinely the correct machine and not the caller itself or another user account. Additionally, there are no checks to ensure that the caller of the extrinsic has previously requested a service from the machine.

We suggest enhancing the *pay_machine_usage* and *get_registration_reward* extrinsics by ensuring the machine is verified within the Peaq network through a registry check, confirming the caller has previously requested a service, and preventing the caller from being the recipient. Moreover, to pay for machine usage, we recommend avoiding the minting of tokens and instead propose using the caller's funds to reward the machine. A safer approach for distributing rewards to online machines would be to use a hook, for example, via *on_initialize*, instead of an extrinsic, to achieve better control over reward distribution.

5.2.9 Incorrect staking requirements could compromise validation security

Attack scenario	**Perform a 51% attack**
Location	peaq-runtime
Tracking	[17]
Attack impact	Attacker may be able to conduct a Sybil attack or a 51% attack
Severity	High
Status	Closed*****

In the parachain_staking pallet, *MinCollatorStake* refers to minimum stake required for any account to be elected as validator for a round. *MinCollatorStake* is currently set too low within the peaq runtime (32_000) compared to krest (50_000 * DOLLARS where DOLLARS = 10 ^ e18).

A low minimum collator stake diminishes the barriers for entry to participate as validators within the network. Malicious actors could, with relatively minimal investment, operate multiple validator nodes, increasing the likelihood of executing a 51% attack. Furthermore, a low stake requirement reduces the cost of acting maliciously for validators, as the potential penalties for slashing are less impactful.

We recommend using DOLLARS, CENTS, MILLICENTS or NANOCENTS whenever a Balance type is involved. Specifically, we advise setting the *MinCollatorStake* in the peaq_runtime to match the levels defined in krest_runtime.

5.2.10 Extrinsic with missing storage deposits could clutter the storage.

Attack scenario	**Disrupt blockchain operation**
Location	pallet-did
Tracking	[18]
Attack impact	An attacker can clutter storage and halt blockchain operation
Severity	High
Status	Closed*****

Storage deposit fees are missing from several extrinsics throughout the Peaq codebase. A malicious entity could call these extrinsics repeatedly and store non-relevant data into the blockchain database to clutter the underlying storage. If the data is never deleted from the blockchain storage, even without malicious interactions, normal operations could clutter the storage over time. This may make the blockchain harder to operate.

As a best practice, every call that writes data into the storage database should be charged a storage deposit or fee.

5.2.11 Usage of old Substrate dependencies

Attack scenario	**Trigger known bugs in Substrate**
Location	peaq-network-node
Tracking	[19]
Attack impact	Attacks may be able to exploit known vulnerabilities
Severity	High
Status	Closed*

Peaq utilizes outdated versions of Substrate, Polkadot, ORML, and other dependencies. Given that the Substrate ecosystem has advanced to version 1.8 [22]. while Peaq uses 0.9.43 [23], this may open Peaq to vulnerabilities patched by Substrate in the versions succeeding Peaq's. We recommend updating to the latest Substrate version. A tool which could potentially simplify the update process is Parity's Polkadot SDK Version Manager [1].

Issue update

As of the 19th of August 2024, SRLabs performed a security review of the asynchronous backing integration within Peaq runtimes.

5.2.12 Incorrect benchmarks for external pallets

Attack scenario	**Send incorrectly benchmarked extrinsics to spam the network**
Location	pallet-sudo, pallet-contracts, pallet-collective, pallet-treasury, pallet-evm, pallet-session, pallet-vesting, pallet-xcm, address-unification, xc-asset-config, staking-coefficient-reward
Tracking	[20]
Attack impact	Attackers may be able to spam transactions
Severity	Medium
Status	Closed*

Peaq depends on a subset of FRAME pallets. The benchmarks for these pallets are done using external runtimes such as substrate-node-template, acala, astar-collator or kilt-parachain, instead of the correct Peaq runtime (i.e., agung, krest, peaq, peaq-dev). Our audit found eleven pallets where benchmarks were sourced from external runtimes.

This may lead to underweight or overweight extrinsics and may harm the credibility of the network. We recommend that all pallet extrinsics, even the Substrate ones, be benchmarked with the actual runtime configuration by including them in the define_*benchmarks!* block.

A best practice example can be found in the Kusama runtime implementation [24].

[1] https://github.com/paritytech/psvm

5.2.13 Import of an insecure randomness algorithm

Attack scenario	**Abuse insecure randomness**
Location	krest, agung, peaq, peaq-dev runtimes
Tracking	[21]
Attack impact	Attackers may be able to exploit any deployed smart contract that utilizes randomness APIs
Severity	Info
Status	Closed*

The source of randomness in the Krest, Agung, Peaq and Peaq-dev runtimes is configured to use the *pallet_insecure_randomness_collective_flip*, implemented in Substrate.

The output of collective flip is highly predictable as it is based on the last eighty-one blocks and should not be used as a true source of randomness. While the usage of collective flip is limited to *pallet_contracts*, which does not indicate a security issue [25] for Peaq, we highly recommend replacing the source of randomness due to the possibility that smart contract developers on the Peaq platform may utilize the insecure randomness functionality, making deployed smart contracts on Peaq vulnerable to exploitation.

6 Evolution suggestions

To ensure that Peaq is secure against known and yet undiscovered threats alike, the auditors recommend considering the evolution suggestions and best practices described in this section.

6.1 Core improvement suggestions to improve security posture

Implement missing validation logic in extrinsics. Incomplete validation logic in extrinsics [12] [13] [9] may lead to severe issues since the blockchain is live. Rigorous implementation, testing and deployment of the necessary validation logic is recommended to secure and affirm the reliability and integrity of the blockchain.

Implement a system to track Machines. Missing logic to accommodate tracking of machines' uptime, availability and services may lead to financial exploitation [15] [16]. It is essential to implement mechanisms to track Machines so that Machine users, owners and operators receive appropriate services and rewards. Additionally, implementing a form of slashing may be beneficial for to deter fraudulent claims and activity.

Benchmark extrinsics appropriately. Benchmarks for foreign pallets must be conducted using the appropriate Peaq runtime instead of a foreign runtime [20]. Additionally, a thorough review of all exposed extrinsics must be conducted to determine if they are benchmarked correctly and if they truly reflect worst case scenarios. Issue #2 [10] may be referred to for an initial subset of extrinsics.

Document and clarify the identity system. Peaq's identity system is highly permissioned and very restrictive, which shields the blockchain from many vulnerabilities. Nevertheless, care should be taken to clarify and document the identity primitives and pallets. Many methods contain duplicated logic, which could be the source of security vulnerabilities in the future. These duplications should be removed, and the code simplified where possible.

Update to the latest Substrate version. Polkadot regularly updates Substrate to fix security issues and improve functionality. It is essential for Peaq's Substrate fork to maintain version parity with upstream Substrate to benefit from security improvements.

Strengthen testing process. The infrastructure with regards to testing is opaque. This will create blind spots in security testing of certain functionalities and might lead to bugs being introduced that would have been caught otherwise. Code coverage is already measured by the Peaq CI/CD tools, but the team should make sure the most critical parts of the system (identity, claims, governance, assets) are sufficiently covered by the existing tests, and take steps to create more tests if it is not the case. Besides, some additional code-comments in the existing tests would greatly help internal and external entities trying to collaborate on the project.

Include concrete examples in documentation. The existing documentation could benefit from improvements as it is very high-level. The team has created exhaustive SDK-level examples, but the project lacks rust-level technical examples. These examples could include processes outlining which extrinsics to use for typical interactions with Peaq, such as creating a child identity, managing portfolios, and adding claims. They could also list the different possibilities available to users when using these extrinsics, as they can be complex and include many options.

6.2 Further recommended best practices

Regularly review the code and continuously fuzz test. Security Research Labs recommends having regular code reviews by security-focused professionals (internal or external to Peaq) to avoid introducing new logic or arithmetic bugs. Continuous fuzz testing can also identify potential vulnerabilities early in the development process. Ideally, Peaq should continuously fuzz their code on each commit made to the codebase. The Polkadot codebase provides a good example of multiple fuzzing harnesses [26]. In addition to these, Security Research Labs has released some example substrate runtime fuzzer harnesses [27].

Regularly update. New releases of Substrate may contain fixes for critical security issues. Since Peaq is a product that heavily relies on Substrate, updating to the latest version as soon as possible whenever a new release is available is recommended. Security Research Labs recommends paying special attention to security fixes, specifically Substrate related ones, as well as setting up a review process for every new main version of Substrate to be incorporated into the update process of Peaq.

Continue improving best practice review process. Finding vulnerabilities is only the start of the remediation process. To ensure that no issue goes unfixed, Security Research Labs recommends continuing to improve upon the team's review process, establishing a set of guidelines and criteria for the review to ensure consistency and standardization.

Avoid forking Substrate, Polkadot and other libraries. Peaq highly depends on forked dependencies such as Frontier, Polkadot, Substrate, ORML, etc. While it may not always be possible to contribute upstream to those components, forking should be avoided in most cases. Depending on those forks makes getting upstream fixes a manual process and harder to maintain.

7 Bibliography

[1] [Online]. Available: https://securityresearchlabs.sharepoint.com/:x:/s/Peaq/ETAWXt0_Pr5CmLQhZXnYGrgBjlGfbe2VD1tiBwqpOfLr2g?rtime=_clT1jRo3Eg.

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

[3] [Online]. Available: https://github.com/peaqnetwork/peaq-pallet-did-audit-srl-2024.

[4] [Online]. Available: https://github.com/peaqnetwork/peaq-pallet-rbac-audit-srl-2024.

[5] [Online]. Available: https://github.com/peaqnetwork/peaq-pallet-mor-audit-srl-2024.

[6] [Online]. Available: https://github.com/peaqnetwork/peaq-storage-pallet-audit-srl-2024.

[7] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node-audit-srl-2024.

[8] [Online]. Available: https://github.com/peaqnetwork/peaq-pallet-transaction-audit-srl-2024.

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

[10] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/2.

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

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

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

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

[15] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/9.

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

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

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

[19] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/7.

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

[21] [Online]. Available: https://github.com/kevin-valerio/srlabs_peaq_audit/issues/1.

[22] [Online]. Available: https://github.com/paritytech/polkadot-sdk/releases/tag/polkadot-v1.8.0.

[23] [Online]. Available: https://github.com/peaqnetwork/peaq-network-node/blob/13b642963ca49b133079c8f99f471cbb8da7732c/Cargo.toml#L80-L212.

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

[25] [Online]. Available: https://paritytech.github.io/polkadot-sdk/master/src/pallet_contracts/lib.rs.html#262-265.

[26] [Online]. Available: https://github.com/paritytech/polkadot-sdk/tree/b13a3187f2b18d1ed1821670f11dc0c70399bb50/polkadot/xcm/xcm-simulator/fuzzer.

[27] [Online]. Available: https://github.com/srlabs/substrate-runtime-fuzzer.

[28] https://github.com/peaqnetwork/peaq-security-assement-reports/blob/dev/2022_11_19_peaq.pdf. [Online].

[29] [Online]. Available: https://github.com/peaqnetwork/peaq-security-assement-reports/blob/dev/2022_12_13_peaq.pdf.

[30] [Online]. Available: https://github.com/peaqnetwork/peaq-security-assement-reports/blob/dev/2023_06_01_peaq.pdf.

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