Security Research Labs

OpenZeppelin runtime configuration template review

Hacking assessment report

V1.1, April 17th, 2024

Haroon Basheer	haroon@srlabs.de
Louis Merlin	louis@srlabs.de
Regina Biro	regina@srlabs.de

Abstract. This work describes the result of a thorough and independent security assurance audit of the OpenZeppelin runtime template 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 audit, OpenZeppelin provided access to relevant documentation and an initial threat model. The code of the OpenZeppelin runtime template was verified to assure that the configuration and logic of the product is resilient to hacking and abuse.

The research team identified and validated the fixes for 1 critical and three high severity issues in configuration related to Cross chain messaging (XCM) and benchmarking for template runtime.

In addition to mitigating the reported issues, Security Research Labs recommends further adherence to security standards in the Polkadot ecosystem through improved documentation of parameters and improved runtime benchmarking. This will accelerate runtime development and security awareness for new developers.

Content

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 4 may not ensure all future code to be bug free.

2 Motivation and scope

OpenZeppelin [1] aims to provide a simplified entry into the Polkadot parachain ecosystem for substrate developers. The runtime configuration template developed by OpenZeppelin strives to provide a common framework and out of the box solution for new developers to bootstrap and expand the Polkadot SDK for parachain development.

In this engagement, the code assurance team focused on finding security vulnerabilities primarily stemming from misconfigurations in the runtime construction library.

Security Research Labs (SRLabs) collaborated with the Parity AppSec team to conduct an in-depth review of the threat model document developed by OpenZeppelin [2]. An independent assessment into the runtime configuration and constructions was also conducted by the SRLabs auditors with their expertise in security auditing for various parachains and supporting secure development of Polkadot-SDK libraries. The OpenZeppelin runtime was assessed for potential security flaws, realistic attack scenarios and adherence to security best practice in the Polkadot ecosystem.

During the assessment of the runtime template, security issues were communicated to the OpenZeppelin and Parity AppSec team through a dedicated Element channel.

3 Baseline Assurance

3.1 Findings summary

During the analysis of the OpenZeppelin runtime template, Security Research Labs identified 1 critical and 3 high-severity issues (4 in total), which are summarized in Table 1.

Issue	Severity	Status
Runtime template waives XCM message delivery fee	Critical	Closed
No XCM delivery fees configured for sibling parachain messages	High	Closed
Incorrect runtime weights for XCM and the Message Queue pallet	High	Closed
Insufficient benchmarking for runtime pallets	High	Closed

Table 1 Code audit issue summary

3.2 Detailed findings

3.2.1 Runtime template waives XCM message delivery fees

Attack scenario	Runtime template waives XCM message delivery fee
Location	runtime/src/xcm_configs.rs
Attack impact	An attacker can cause network congestion causing delay in message deliveries or discards and storage exhaustions

Severity	Critical
Status	Closed [3]

To avoid congestion and spamming of messages in the network via XCM, a fee is charged for message delivery to ensure fairness and optimal message delivery time amongst all the participants. This fee is configured in the runtime construct through use of *FeeManager* type.

In the template for XCM runtime configuration for OpenZeppelin, have set the *FeeManager* [4] to the Rust unit type "()". In the Polkadot-SDK, this implementation effectively will waive the delivery fees [5].

Thereby all fee-based congestion control mechanisms become ineffective since the fees are not actually charged. An attacker can cause congestion, possibly leading to long delays in message delivery, storage exhaustion and/or dropping of messages.

We recommend implementing a fee for XCM message delivery and follow the fix implemented in the runtime update v1.3 [6].

3.2.2 No XCM delivery fees configured for sibling parachain messages

Attack scenario	No XCM delivery fees configured for sibling parachain messages
Location	runtime/src/lib.rs
Attack impact	Attackers may send spam messages across chains without paying a fee
Severity	High
Status	Closed [7]

To ensure fairness among users and prevent spamming of messages across chains, an adequate fee mechanism must be implemented for sending the XCM message.

Currently, there are no fees charged for delivering XCM messages across parachains. In the template runtime configuration, this is configured through *PriceForSiblingDelivery* by *NoPriceForMessageDelivery*.

Attackers may send spam messages across chains without paying a fee. Excessive messages could lead to XCM queue size exhaustion by excessive storage usage until messages are delivered. This could also lead to delays in message delivery for other users.

Charge adequate message delivery fees in the runtime configuration template. To prevent excessive delivery times and storage exhaustion, an exponential fee mechanism should be used as configured in Kusama [8].

3.2.3 Incorrect runtime weights for XCM and the Message Queue pallet

Attack scenario	Incorrect Runtime weights for XCM and the Message Queue pallet
Location	runtime/src/xcm_configs.rs

Attack impact	As these pallet extrinsic weights are not dependent on the actual runtime configuration, this could lead to underweight extrinsics
Severity	High
Status	Closed [9]

OpenZeppelin's runtime template constructs runtime logics with FRAME pallets in its configuration. Appropriate benchmarking is required for these pallets to ensure its actual runtime performance. This will effectively transform into charging adequate fee for extrinsics execution.

For example, the runtime weights for pallet XCM are configured using *TestWeightInfo* [10] Similarly for Generalized Message Queue runtime weights are configured to be Zero as type *WeightInfo = ()* [11].

As these pallet extrinsic weights are not dependent on the actual runtime configuration, this could lead to underweighted extrinsics. Setting the weights to () effectively make it a zero-cost execution for extrinsic. Both scenarios lead to attackers spamming and bloating the network storage for free.

All pallet extrinsics, even the Substrate ones, should be benchmarked with the actual runtime configuration by including them in *define_benchmarks!* block. A best practice example can be found in the Kusama runtime implementation [12]

3.2.4 Insufficient benchmarking for runtime pallets

Attack scenario	Insufficient benchmarking for runtime pallets
Location	runtime/src.rs
Attack impact	Missing or incorrect runtime benchmarks result in overweight or underweight extrinsics. This can lead to low-effort attacks such as spamming, storage bloating, and block stalling
Severity	High
Status	Closed [13]

Pallets with extrinsics must have an accurate weight function which factors in storage, database access and computation. The reuse of the substrate template and failing to benchmark weights may result in a mismatch of computational requirements and the cost of execution.

Some pallets use pre-defined weights in the runtime configuration but are not included in the runtime benchmarks [14]. The pallets affected are:

- *cumulus_pallet_parachain_system*,
- *parachain_info,pallet_proxy*,
- *pallet_utility,pallet_multisig*,
- *pallet_transaction_payment,*

- *pallet_xcm*

Additionally, the benchmarks for all the pallets are using *SubstrateWeights* [15] which were benchmarked according to the substrate-node template, instead of the actual template runtime.

Excluding pallets with configured weights from benchmarking and using the substrate-node template runtime weights may result in overweight or underweight extrinsics in the runtime environment. This potentially leads to low-effort attacks such as spamming, storage bloating, and block stalling when invoking extrinsics.

Include the above-mentioned pallets in the *try_benchmarks!* macro for appropriate runtime benchmarks. Fix all the defaults benchmarking with actual runtime, please refer to the Glutton-Kusama parachain [16] as a best practice example.

4 Evolution suggestions

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

4.1 Implement stringent filtering for proxy calls

The *frame_system::Config* implements a *NormalFilter* [17] for the proxy account functionality. This filtering implementation effectively safeguard accounts from being unable to access their unreserved funds, which could be triggered through proxy function calls such as *kill_pure* and *create_pure*.

The filtering logic could be further enhanced by adding another function call *remove_proxies* [18], which assists in removing proxy accounts created through the *create_pure* function. The *remove_proxies* function call also has the potential to cause an inability to access unreserved funds, as highlighted in the inline comment.

This is fixed and strict proxy filtering is implemented [19].

4.2 Adhere to benchmarking best practices standard to the polkadot-sdk

Ensure correct and adequate benchmarking for all the pallets configured in the runtime. Failure to do so significantly undermines the operation of the blockchain. It is imperative that the developers consult the Polkadot-SDK's knowledge base to ensure proper benchmarking for all the pallets in the runtime. This also prevents exposing the parachain to low-effort attacks such as spamming and bloating of chain storage.

The suggestion is taken into consideration in the fix for XCM weights and runtime benchmarking issues.

4.3 Improve documentation of runtime parameters

As runtime templates are meant to help with bootstrapping new developers to accelerated development, it's critical to reason-about and document hardcoded parameters, especially when configuring such parameters has a security relevance. This will help developers gain a deeper understanding of the inner workings of the parachain ecosystem and effectively navigate its complexities through the runtime template. For instance, SS58Prefix configuration in the *frame_system::Config* is

hardcoded to 42 [20]. Expanding on the rationale for the usage of value 42 can be further explained using the ss58-registry [21].

It is recommended to improve the documentation and inline commenting for all the parameters in the runtime template to make the developers aware of the rationale behind the value. This will also help the developers in making informed decisions before tweaking the parameters to suit their business logic, thereby improving their security posture.

This is fixed and the comment to the parameters is fixed [22].

4.4 Remove references to archived repositories

A code comment [23] that explains constraints for proxy account filtering points to a read only archived substrate repository, which might lead to an incorrect knowledge base and reference. Update the comments with latest commit hash to the Polkadot-SDK [24] to ensure accurate and up-to date information.

This is fixed together with Section 4.1's suggestion.

4.5 Improve threat model created for the runtime template

The threat document shared by OpenZeppelin [2] highlights generic threats without considering attacks and risks pertaining to runtime misconfigurations. The list of topics such as Assets, Actors, Entry Points, Trust Levels in the threat model does not provide insights or connections to the runtime threats and attacks. For example, some typical runtime-specific threats such as storage bloating, message spamming through underweight extrinsics are not considered.

Only generic and abstract issues were enumerated such as unsafe arithmetic underflows/overflows and supply chain threats through external malicious dependencies affecting parachains at large.

Consider including hacking damage indicators that highlight high-risk threats and low-effort attacks to the runtime. A holistic threat assessment specific for the runtime template should include all threats and attacks that concern the configuration of the pallets in the target runtime template.

5 Bibliography

[1] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/tree/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094.

[2] [Online]. Available: https://docs.google.com/document/d/1Hwz1sxp73RLBnsnbYMDGSWyS6Kj7p7lpjumgCkHifqg/edit?usp=sharing.

[3] [Online]. Available: https://github.com/OpenZeppelin/polkadot-generic-runtime-template/pull/159.

[4] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/xcm_config.rs#L153.

[5] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/64660ee8d2e0d0ab4be3d416342463aaa6e168bd/polkadot/xcm/xcm-executor/src/traits/fee_manager.rs#L55-L57.

[6] [Online]. Available: https://github.com/polkadot-fellows/runtimes/pull/87.

[7] [Online]. Available: https://github.com/OpenZeppelin/polkadot-generic-runtime-template/pull/156/files.

[8] [Online]. Available: https://github.com/polkadot-fellows/runtimes/pull/87.

[9] [Online]. Available: https://github.com/OpenZeppelin/polkadot-generic-runtime-template/pull/153.

[10] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/xcm_config.rs#L216.

[11] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/lib.rs#L556.

[12] [Online]. Available: https://github.com/paritytech/polkadot/blob/01fd49a7fafa01f133e2dec538a2ef7c697a26aa/runtime/kusama/src/lib.rs#L1578-L1587.

[13] [Online]. Available: https://github.com/OpenZeppelin/polkadot-generic-runtime-template/pull/149/files#diff-0ec06ea58bd455f09ce6b3bb4c2c1c0d37bda51c1e1be2151c560c9c973959ec.

[14] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/lib.rs#L711.

[15] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/lib.rs#L525.

[16] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/360581ffedd048262177ddc135d66cec455b959a/system-parachains/gluttons/glutton-kusama/src/lib.rs#L219.

[17] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/lib.rs#L283C1-L295C2.

[18] [Online]. Available: https://docs.rs/pallet-proxy/latest/src/pallet_proxy/lib.rs.html#256-268.

[19] [Online]. Available: https://github.com/OpenZeppelin/polkadot-generic-runtime-template/pull/140/files.

[20] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/lib.rs#L279.

[21] [Online]. Available: https://github.com/paritytech/ss58-registry/blob/685be59c05454ce0baf6874d237c6c40abe3d53b/ss58-registry.json#L391-L399.

[22] [Online]. Available: https://github.com/OpenZeppelin/polkadot-generic-runtime-template/pull/151/files.

[23] [Online]. Available: https://github.com/OpenZeppelin/polkadot-runtime-template/blob/c9a2c76a9db2e114eecaeba07195f9c2bdfaa094/runtime/src/lib.rs#L287.

[24] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/3c6ebd9e9bfda58f199cba6ec3023e0d12d6b506/substrate/frame/proxy/src/lib.rs#L260.