Kagome Follow-up Security Assurance

Threat model and hacking assessment report

v1.0 FINAL, April 25, 2025

Prepared for:
Quadrivium

- Public, 46 pages -

Content

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 3.

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

Version:	v1.0 FINAL
Prepared For:	Quadrivium
Date:	April 25, 2025
Prepared By:	Marc Heuse Gabriel Arnautu Nils Ollrogge Ivo Mueller Aarnav Bos Louis Merlin

Timeline

Security Research Labs started the follow-up Kagome source code security assessment in September. The analysis took three months.

Date	Event
November 17, 2023	Initial engagement
May 2, 2024	Report for the baseline security check delivered
September 2, 2024	Follow-up engagement kick-off
April 25, 2025	Report for the follow-up security check delivered

Table 1: Audit timeline

Integrity Notice

1 Executive summary

1.1 Engagement overview

Security Research Labs performed a follow-up security assurance audit of the Kagome Polkadot host implementation by Quadrivium. This report outlines the audit results, focusing on the resilience of the implementation against hacking and abuse scenarios. Since 2019, Security Research Labs has provided specialized audit services within the Polkadot ecosystem.

During this audit, Quadrivium granted access to relevant documentation and actively supported the research team. The auditors examined the design, configuration and available source code of the Kagome Polkadot host implementation in the v0.9.5 [1] release, which was added since the initial baseline security assurance audit conducted by Security Research Labs for the v0.9.3 release.

This audit focused on assessing Kagome's codebase for resilience against hacking and abuse scenarios. Security Research Labs prioritized reviewing critical functionalities and conducting thorough security tests of the newly added features since the previous audit to ensure the correctness and robustness of the Kagome Polkadot host implementation. The auditors collaborated closely with the Quadrivium team, utilizing full access to source code and documentation to perform a rigorous assessment.

1.2 Observations and Risk

The research team discovered several issues, ranging from high-severity vulnerabilities to informational-level concerns. Key concerns included security issues related to memory safety, the software development lifecycle and differential analysis compared to the Polkadot host implementation in Rust. These issues were categorized based on their potential impact on system integrity and exploitation likelihood.

Quadrivium has acknowledged these findings and is actively collaborating with the audit team to resolve them. Remediation efforts are underway, with some vulnerabilities already fixed, prioritizing the most critical vulnerabilities to mitigate immediate risks, followed by measures to address the lower-severity issues to enhance the overall security posture.

1.3 Recommendations

Security Research Labs recommends that Quadrivium mitigate the remaining open issues and increase the project's security maturity by establishing documented processes for implementing test cases and providing sufficient documentation to ensure parity with the Polkadot implementation in Rust. Additionally, we encourage the development team to create well-structured pull requests with well-phrased commits to improve code review efficiency and maintainability. Finally, continuous node and validator tests with sanitizers enabled are highly recommended.

2 Evolution suggestions

Quadrivium, in collaboration with Security Research Labs, has made progress in improving their security metrics and has implemented some of the security recommendations outlined in the previous audit to enhance their code security measures. However, not all recommendations have been fully addressed. To ensure that Kagome is secure against both known and yet undiscovered threats, we recommend addressing the remaining evolution suggestions and best practices described in this section.

2.1 Development lifecycle improvement suggestions

We recommend enhancing the development practices of Kagome by implementing the following recommendations:

Pull request management. Developers should manage pull requests (PRs) effectively to maintain code quality and ensure efficient reviews. PRs should clearly describe the purpose and scope of the changes, be kept as small as possible to improve readability and ease of review and maintain a clean and logical commit history. For example, developers should ensure that pull request changing code indentation are separate from feature changes. Well-documented and well-structured PRs make it easier for reviewers to understand the changes, identify potential issues, and ensure the code aligns with project goals and specifications. Incorporating these practices into the development workflow will lead to a maintainable codebase and increased efficiency during reviews.

Open-source contribution incentive. The Polkadot ecosystem actively encourages open-source contributions through various initiatives, such as grants and funding programs. For instance, the Web3 Foundation Grants Program provides funding for software development and research efforts related to Polkadot. Quadrivium could implement similar incentives for Kagome to attract external developers, enhance community engagement, and accelerate project development. Establishing a structured grants program or offering bounties for specific tasks could motivate contributors to participate, leading to a more robust and dynamic codebase.

2.2 Secure development improvement suggestions

We recommend to further strengthen the security of Kagome by implementing the following recommendations:

Regular codebase updates. Quadrivium should regularly update the Kagome implementation to maintain parity with the latest Polkadot SDK releases and ensure security. New releases often include critical security fixes and new features, which should be integrated promptly to maintain compatibility and robustness.

Use static analysis. The development team should use static analysis tools to detect security flaws and improve the codebase's security. These tools, such as Clang Static Analyzer [2], Cppcheck [3] and Semgrep [4] for the C/C++ ecosystem, analyze the code without executing it, identifying vulnerabilities, programming errors, and compliance issues early in the development process. This proactive approach helps developers address potential security issues before they reach production, ensuring a more secure and reliable codebase.

Perform dynamic analysis. The development team should use fuzzing and build targeted fuzzing harnesses for critical components to identify security vulnerabilities. By combining fuzzing with application-specific invariants, they can also detect business logic issues. Additionally, developers should implement security-focused unit tests as part of their development best practices. These proactive measures help uncover vulnerabilities and logic flaws early, ensuring a more secure and reliable codebase.

Launch a bounty program. Quadrivium should establish a bounty program to encourage external researchers to discover and report security vulnerabilities, enhancing code security. By offering incentives, such programs motivate individuals to responsibly report vulnerabilities to Kagome rather than exploiting them. This approach not only broadens the pool of people actively searching for security flaws but also fosters a collaborative security environment, helping to identify and address issues more quickly and effectively.

2.3 Address currently open security issues

We advise Quadrivium to address known security issues promptly to prevent attackers from exploiting them – even if an open issue has a limited impact, an attacker might use it as part of their exploitation chain, which may have a more severe impact on Kagome.

2.4 Further recommended best practices

Perform threat modeling. Performing threat modeling for all new features and major updates before coding is crucial for better code security. This practice allows developers to identify potential security threats and vulnerabilities early in the design phase, enabling them to implement appropriate mitigations from the outset. Including the threat model in the pull request description ensures that the entire team is aware of the identified risks and the measures taken to address them, promoting a proactive security culture and enhancing the overall robustness of the codebase. Additionally, it helps the audit team to identify gaps in the threat model and focus their assessment.

Create an incident response plan. Developing a comprehensive incident response plan to address potential security breaches is vital for maintaining code security and organizational resilience. This plan should include detailed procedures for responding to various scenarios, such as compromised developers or exploited blockchain vulnerabilities, ensuring quick and effective mitigation of threats. By having a well-defined response strategy, organizations can minimize the impact of security incidents, protect sensitive data, and maintain trust with users and stakeholders.

3 Motivation and scope

Blockchains evolve in a trustless and decentralized environment, which by its own nature could lead to security issues. Ensuring availability and integrity is a priority for Kagome as it aims for a feature-identical implementation of the Polkadot host in C++20. 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 Quadrivium team can then integrate into their own threat modelling and development process to further enhance the security of the product.

Kagome is a blockchain node built to integrate seamlessly with the Polkadot network. With a goal of providing further accessibility and utility outside the Rust ecosystem, Kagome code is written in C++20, a low-level, general-purpose, cross-platform programming language. Polkadot SDK-based chains primarily rely on three key technologies: a WebAssembly (WASM) based runtime, decentralized communication via libp2p, and a block production engine.

The Kagome host implementation executes the WASM runtime, specifically the relay chain runtime. The runtime us the on-chain logic that defines the rules and state transitions of a blockchain, such as consensus mechanism, staking, governance, and other core functionalities. The 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.

The core business logic of Kagome is to implement a node compatible with other Polkadot host implementations, in line with the Polkadot specification but as an alternative implementation so that the Polkadot ecosystem does not depend on a single client.

Security Research Labs collaborated with the Quadrivium team to create an overview containing the components in scope and their audit priority. The in-scope components and their assigned priorities are reflected in table 2.

Repository	Priority	Component(s)
https://github.com/qdrvm/kagome/tree/v0.9.5	High	Asynchronous backing
	High	Elastic scaling
	High	Grid & cluster topologies
	High	Secure validator mode
	High	Validation protocol upgrade
	Medium	Disabled validators mechanism
	Medium	SCALE codec
	Low	Availability-recovery from systematic chunks
	Low	DHT Authority discovery record creation time
	Low	Core runtime API changes for Multi-Block-Migrations

Table 2: In-scope Kagome‘s components with audit priority

4 Methodology

This report details the continuous security assurance results for the Kagome Polkadot host implementation with the aim of creating transparency in four steps: threat modeling, security design coverage checks, implementation baseline check and finally remediation support. We applied the following four steps methodology when performing feature and PR reviews.

4.1 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk for Kagome nodes. 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 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 3.

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

Table 3: 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 Kagome's Polkadot host functionality or reputation.
- Medium: Risk scenarios pose a considerable threat to Kagome's Polkadot host functionality or reputation.
- High: Risk scenarios pose an existential threat to Kagome's Polkadot host functionality or reputation.

Damage and Hacking Value are divided according to table 4.

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

Table 4: Risk measurement scale

After applying the framework to the Kagome 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 in the context of the Kagome Polkadot host implementation involve the exposure of sensitive information related to the blockchain network, validators, and users. For example, attackers could exploit information leaks to identify private keys, validator participation patterns, or node-specific data. Such leaks could allow malicious actors to compromise nodes, disrupt consensus, or illegitimately claim assets (e.g., native tokens) associated with the network. Protecting sensitive data is crucial to maintaining the confidentiality and integrity of the Kagome ecosystem.

Integrity:

Integrity threat scenarios threaten to disrupt the functionality of the entire network by undermining or bypassing the rules that ensure that Kagome node transactions/operations are fair and equal for each participant. Undermining Kagome's integrity often comes with a high monetary incentive, like for example, if an attacker can double spend or mint tokens for themselves. Other threat scenarios do not yield an immediate monetary reward, but rather, could threaten to damage Kagome's functionality and, in turn, its reputation. For example, insufficient consistency with the Polkadot specification would violate the core promise of feature-for-feature compatibility with other Polkadot host implementations.

Availability:

Availability threat scenarios refer to compromising the availability of data stored by the Kagome node as well as the availability of the network itself 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 provides a high-level overview of the hacking risks concerning Kagome 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 *SRLabs-Kagome_Audit-Threat_Model-2023*. This list can serve as a starting point to the Quadrivium 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 Kagome, the auditors attributed the most hacking value to the integrity class of threats. Since the efforts required to exploit this kind of issue are considered lower, we identified threat scenarios to the integrity of Kagome as of the highest risk category. Undermining the integrity of the Kagome node means making unauthorized modifications to the system. Some of the scenarios can have a direct effect on the financials of the system. This can include market manipulation, gaining tokens for free, or as a vault stealing collateral without repercussions.

Security promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confidentiality	Medium	- Steal private keys from a compromised node to access and transfer tokens from associated accounts	High	- Decrypt private information through bad usage of cryptographic protocol
Integrity	High	- Double spend tokens via getting the clients to accept a different chain - Undermine consensus mechanism to split chain - Tamper / manipulate blockchain history to invalidate transactions (e.g. a voting result)	Medium	- Double spend through a nothing-at-stake attack - Modify critical business data by exploiting logic error in the code - Transaction malleability attack - Consensus split
Availability	High	- Conduct (D)DoS attacks to disrupt network connectivity and operations - Undermine block production or consensus mechanism	Low	- DoS validator nodes - Transaction spam - Find memory leaks

Table 5: Risk overview. The threats for Kagome's Polkadot host implementation were classified using the CIA security triad model, mapping threats to the areas: (1) Confidentiality, (2) Integrity, and (3) Availability.

4.2 Implementation check

As a third step, the current Kagome host implementation was tested for openings whereby any of the defined hacking scenarios could be executed.

To effectively review the Kagome codebase, we derived our code review strategy based on the threat model that we established as the first step. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category, as outlined in Chapter 4.1.

Prioritizing by risk, the code 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 namespace and the deployment configuration.

2. Identified viable strategies for the code review. Manual code audits, fuzz testing, and manual tests were performed where appropriate.

3. Ensured the code did not 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 reported any vulnerability that was discovered to the development team along with suggestions around mitigations.

We carried out a hybrid strategy utilizing a combination of code review, static test is and dynamic tests (e.g., fuzz testing) to assess the security of the Kagome codebase.

While static testing, fuzz testing and dynamic tests establish a baseline assurance, the focus of this audit was a manual code review of the Kagome codebase to identify logic bugs, design flaws, and best practice deviations. We reviewed version 0.9.5 of the Kagome repository [1] up to commit 2433e35 *from the 30st of August 2024.* The approach of the review was to trace the intended functionality of the 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 Kagome 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. 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., type confusion, integer overflow) in the method under test. In Kagome's case we focused our fuzzing effort on the SCALE codec implementation.

4.3 Remediation support

The final step is supporting Quadrivium 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.

During the audit, findings were shared via a private GitHub repository [5]. Additionally, a private Telegram channel for asynchronous communication and status updates was used. Bi-weekly jour fixe meetings were held to provide detailed updates and address open questions.

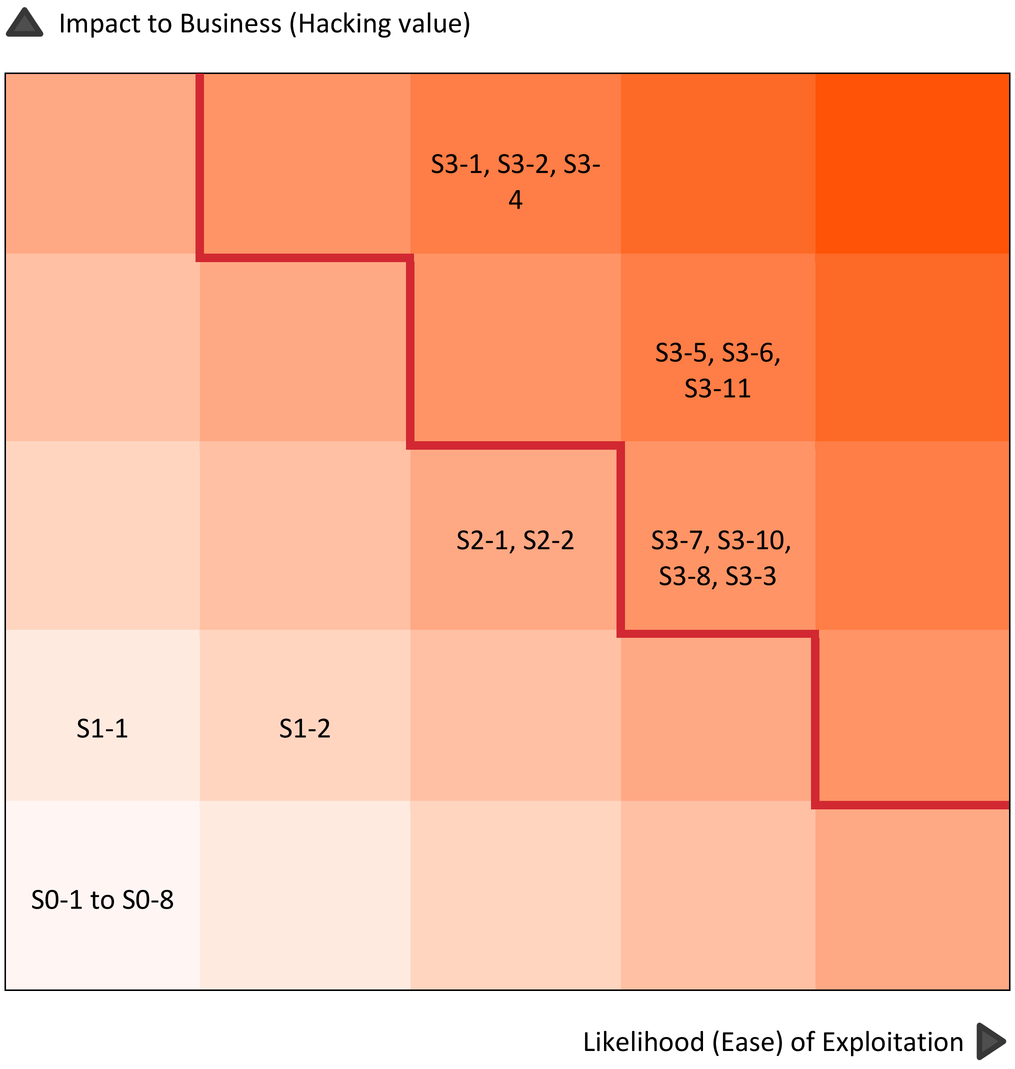
5 Findings summary

We identified 22 issues during our analysis of the modules in scope in the Kagome codebase that enabled some of the attacks outlined in table 5. In summary, 10 high severity, 2 medium severity, 2 low severity and 8 info level issues were found. An overview of all findings can be found in table 6.

High	10
Medium	2
Low	2
Informational	8
Total Issues	**22**

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

ID	Issue	Severity	Status
S3-1 [6]	Incorrect implementation of `is_potential_spam` check	High	Mitigated [7]
S3-2 [8]	Behavioral differences in dispute participation with disabled validators	High	Mitigated [9]
S3-3 [10]	Mismatch in SCALE decoding of `CompactInteger`	High	Mitigated [11]
S3-4 [12]	Missing offchain disabled validator state	High	Mitigated [13]
S3-5 [14]	Kagome failing with UBSAN or TSAN enabled	High	Risk accepted
S3-6 [15]	Heap Use-After-Free in Kagome Node when running as validator	High	Mitigated
S3-7 [16]	Numerous TODOs in Kagome Code - some untracked - Affecting Security and Stability	High	Partially mitigated
S3-8 [17]	Insufficient codebase comments	High	Partially mitigated
S3-10 [18]	Lack of BEEFY tests leads to potential implementation differences	High	Mitigated [19, 20, 21, 22]
S3-11 [10]	Kagome RPC crashes	High	Mitigated [23, 24, 25]
S2-1 [26]	Missing Reputation Penalties in Peer Request Handling	Medium	Mitigated [27]
S2-2 [28]	Missing Failure Conditions in Assignment Certificate Verification	Medium	Mitigated [29, 30, 31]
S1-1 [32]	Secure Validator Mode does not support clone instead of fork for new job processes	Low	Mitigated [33]
S1-2 [34]	Insecure input validation using `debug_assert` in unsafe functions of erasure-coding-crust module	Low	Mitigated [35]
S0-1 [36]	Prospective parachains: Missing test introduced with elastic scaling	Info	Mitigated [37, 38]
S0-2 [39]	Missing support for new runtime API function candidates_pending_availability	Info	Mitigated [40]
S0-3 [41]	Prospective parachains: `FragmentTree_checkBackableQueryMultipleCandidates` function missing some testcases.	Info	Mitigated [42]
S0-4 [43]	Prospective parachains: Backing tests missing	Info	Mitigated [44, 45]
S0-5 [46]	Grid topology: missing tests	Info	Mitigated

S0-6 [47]	Missing tests for DHT Authority discovery record creation time	Info	Mitigated [48]
S0-7 [49]	Systematic chunks recovery: missing tests	Info	Mitigated [50]
S0-8 [51]	Lack of Parachain Tests in Kagome Leads to Unverified Implementation Consistency with Polkadot	Info	Risk accepted

Table 6: Findings overview

6 Detailed findings

6.1 S3-1: Incorrect implementation of is_potential_spam check

Tracking	**[6]**
Severity	High
Status	Mitigated [7]

Issue description
The Polkadot-SDK reference implementation includes an `is_potential_spam` function that checks whether a dispute for a given block candidate could be spam.

In the Polkadot-SDK reference implementation [52], the check is written as follows:

```
let ignore_disabled = !is_confirmed && all_invalid_votes_disabled;

(is_disputed && !is_included && !is_backed && !is_confirmed) || ignore_disabled
```

On the Kagome side, the check is implemented [53] slightly differently:

```
auto is_potential_spam = is_disputed  //
                        and not is_included and not is_backed
                        and not is_confirmed and not is_postponed;
```

In Kagome, `is_postponed` is set to true if the dispute was initiated exclusively by disabled validators. Therefore, the check could also be written as:

```
auto is_potential_spam = is_disputed  //
                        and not is_included and not is_backed
                        and not is_confirmed and not all_invalid_votes_disabled;
```

When comparing the checks the Kagome check differs from the one of the Polkadot-SDK reference implementation.

Risk
This difference in how spam disputes are evaluated leads to divergent behaviors between Kagome and the Polkadot-SDK in specific scenarios. For example, in an unconfirmed dispute with votes solely from invalid validators. Assuming `is_included` and `is_backed` are `false`, the Kagome check would return `false`, while the Polkadot-SDK implementation would return `true`.

Mitigation suggestion
To mitigate this risk, we recommend aligning Kagome's `is_potential_spam` check with the Polkadot-SDK implementation to ensure consistent behavior.

6.2 S3-2: Behavioral differences in dispute participation with disabled validators

Tracking	**[8]**
Severity	High
Status	Mitigated [9]

Issue description
In the Polkadot-SDK reference implementation, a confirmed dispute involving a disabled validator should still be imported, and participation should occur, as asserted by this test [54].

In Kagome [55], participation is decided upon based on the following check:

```
if (own_vote_missing                    //
    and is_disputed and not is_postponed //
    and allow_participation) {
```

Here, the `is_postponed` variable is set to true when a dispute has only votes from invalid validators for a given vote type.

As a result, when a confirmed dispute involves only disabled validators, a validator using the Kagome host implementation would not participate in the dispute because not `is_postponed == false`.

Risk
This difference in dispute participation behavior creates a mismatch between Kagome and Polkadot-SDK implementations during confirmed disputes with only disabled validators. Such a discrepancy allows a malicious PVF to distinguish between the Kagome and Polkadot-SDK host implementations, potentially enabling the targeted triggering of disputes (and slashing) of honest validators running different host implementations.

Mitigation suggestion
To mitigate this risk, we recommend aligning Kagome's dispute participation logic with the Polkadot-SDK implementation to ensure consistent behavior.

6.3 S3-3: Mismatch in SCALE decoding of CompactInteger

Tracking	**[10]**
Severity	High
Status	Mitigated [11]

Issue description

Kagome's SCALE codec implementation **scale-codec-cpp** has a mismatch when decoding `CompactInteger` when compared with the reference implementation **parity-scale-codec**.

When decoding the following input as a `CompactInteger`: `[0x03, 0x01, 0x00, 0x00, 0x02, 0x02, 0x00, 0x1c], the` **parity-scale-codec will fail** with the following error:

```
"parity_scale_codec::compact::Compact<u64>"::Err(
    Error {
        cause: None,
        desc: "out of range decoding Compact<u64>",
    },
)
```

However, Kagome's **scale-codec-cpp** will successfully decode the input to `33554433 ([0x06, 0x00, 0x00, 0x08])`.

Due to the usage of `CompactInteger` to encode the length of a dynamic collection, all dynamic collection data types are affected. This includes:

- `DynamicCollection`
- `ResizableCollection`
- `ExtensibleBackCollection`
- `RandomExtensibleCollection`

Risk

Invalid decoding of `CompactInteger` may lead to data corruption and may significantly affect the integrity of the blockchain.

Mitigation recommendation

Implement `CompactInteger` decoding correctly.

6.4 S3-4: Missing offchain disabled validator state

Tracking	**[12]**
Severity	High
Status	Mitigated [13]

Issue description
The Kagome host implementation does not maintain offchain storage of disabled validators, which is necessary to prevent past-era dispute spam. This feature is specified in the host implementers guide [56] and has been implemented [57] by the Polkadot-SDK reference implementation. The lack of this protection leaves the system vulnerable to attacks that can exploit past eras.

Risk
A malicious validator could exploit this gap by disputing all valid block candidates from previous eras, disrupting consensus and causing potential downtime or delays in block finalization.

Mitigation recommendation
To mitigate this risk, we recommend implementing offchain disabled validator tracking as recommended. This solution would strengthen protection against past-era disputes, ensuring compliance with the implementers guide. It may require additional resources to manage offchain storage but would provide crucial defense. Not implementing this could avoid the extra complexity but leaves the system exposed to potential attacks from compromised validators.

6.5 S3-5: Kagome failing with UBSAN or TSAN enabled

Tracking	**[14]**
Severity	High
Status	Acknowledged

Issue description
The Kagome node and test encountered repeated test failures when either Undefined Behavior Sanitizer (UBSAN) or Thread Sanitizer were enabled. In the current Continuous Integration (CI) setup, these sanitizers attempt to report issues without halting the tests, which lead to incomplete and unreliable reporting.

Risk
Undefined behavior and unsafe threaded data accesses are present in the Kagome code which can impact the security and reliability of Kagome.

Mitigation recommendation
Enforcing the `-fno-sanitize-recover=all` flag in the CI setup will ensure that any undefined behavior or thread-related issues immediately stop test execution, thereby preventing any undefined behavior from influencing further test outcomes. Additionally, the Kagome node software when compiled for TSAN or UBSAN should also receive this flag abort on errors during running.

6.6 S3-6: Heap Use-After-Free in Kagome Node when running as validator

Tracking	**[15]**
Severity	High
Status	Mitigated

Issue description
When running Kagome as a validator, a heap-use-after-free (UAF) occurs after some time. This issue was not manually triggered but occurred during standard operations of the validator with ASAN enabled.

Risk
This vulnerability poses a moderate to high risk. In standard operation without ASAN enabled, the node is unlikely to crash; however, a UAF may lead to undefined behavior within the affected thread. The worst-case scenario can lead to a write-what-where condition

Mitigation recommendation
We recommend fixing the UAF vulnerability. Additionally, the validator should be compiled with different sanitizers and run for several days each to identify new and lurking issues. Enforcing `the -fno-sanitize-recover=all` flag in the CI setup will further ensure that any address, undefined behavior, or thread-related issues immediately stop test execution, thereby preventing errors from influencing further test outcomes.

6.7 S3-7: Numerous TODOs in Kagome code

Tracking	**[16]**
Severity	High
Status	Acknowledged

Issue description
The Kagome codebase contains over 100 TODO comments of which a larger number are not linked to active issues in the project's issue tracking system. The high number of TODOs may relate to missing important features or security enhancements.
The lack of linkage results in poor traceability and accountability for pending tasks.
This can significantly impact the overall security and stability of the Kagome implementation as part of the Polkadot ecosystem.

Risk
The volume of over 100 TODOs exacerbates this risk by complicating prioritization and resolution efforts, potentially leading to a backlog that can impede ongoing development and maintenance.
The main risk associated with untracked TODOs is the potential for critical improvements, optimizations, or security patches to be overlooked or delayed. Without proper tracking, these TODOs may represent unresolved vulnerabilities or areas of the code that require refinement for optimal performance and security.

Mitigation recommendation
We recommend focusing on TODOs for a specified amount of time to reduce their number and either close as unneeded (and document this) or implement them.

To address the risks posed by untracked TODOs in the Kagome codebase we recommend linking all untracked TODOs to active issues in the Issue Tracking System.
Additionally a policy should be established that requires all new TODOs to be accompanied by an issue in the tracking system to ensure ongoing accountability.

6.8 S3-8: Insufficient codebase comments

Tracking	**[17]**
Severity	High
Status	Acknowledged

Issue description

The Kagome project's codebase has a significant imbalance in the distribution of code comments, with many areas lacking adequate commentary. This discrepancy leads to uneven understanding and maintainability across different parts of the code. As an example, the BABE module has one comment for every 41 lines. In comparison, the Polkadot implementation of BABE has a ratio of 1:6.

Risk

The absence of comments in critical areas can result in increased difficulty for developers in understanding and modifying the code. This lack of clarity can lead to a higher potential for errors, inefficient coding practices, and challenges in future maintenance and updates. Furthermore, new contributors or team members may face a steep learning curve, potentially slowing down development progress.

Mitigation recommendation

Depending on code complexity, established recommendations for line-to-code rations vary between 1:10 and 1:25. When Quadrivium starts adding comments to the code, functions with high cyclomatic complexity or critical functionality should be prioritized. Also links to the Polkadot-SDK reference implementation should be added to the comments.

6.9 S3-10: Lack of BEEFY tests leads to potential implementation differences

Tracking	**[18]**
Severity	High
Status	Mitigated [19, 20, 21, 22]

Issue description
Kagome omits implementing BEEFY tests from Polkadot. This exclusion raises concerns regarding the verification of its implementation's consistency and compatibility with Polkadot's ecosystem. BEEFY tests are crucial for ensuring that implementations like Kagome can integrate seamlessly with the consensus model of the Polkadot ecosystem.

Risk
The absence of BEEFY tests in Kagome's testing framework poses a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation. This gap undermines the confidence in Kagome's ability to support the Polkadot ecosystem's consensus architecture effectively.

Mitigation recommendation
To address this issue, we recommend integrating existing BEEFY tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing BEEFY tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

6.10 S3-11: Kagome RPC crashes

Tracking	**[10]**
Severity	High
Status	Mitigated [23, 24, 25]

Issue description
During testing of the Kagome node's RPC interface, we identified three distinct crashes that can be reliably triggered. These occur when a client connects to port 9944 and sends specially crafted, malformed requests.

Risk
These crashes present a high risk as they enable remote, unauthenticated attackers to perform denial-of-service (DoS) attacks. An attacker could exploit this to repeatedly crash the node, potentially disrupting network participation, reducing availability, and impacting reliability.

Mitigation recommendation
We recommend identifying the root causes of these crashes and addressing them to improve the stability of the Kagome node and strengthen the trust of the Polkadot ecosystem in it.

6.11 S2-1: Missing reputation penalties in peer request handling

Tracking	**[26]**
Severity	Medium
Status	Mitigated [27]

Issue description
The Polkadot SDK penalizes invalid peer requests as part of the statement distribution protocol. For example, when handling requests for candidates, the Polkadot SDK reduces [58] a peer's reputation if it is not allowed to request the candidate. While the Kagome implementation correctly performs the necessary checks, it does not [59] send messages to reduce the reputation of the offending peer.

Such error-handling network messages are missing in various other places as well, for example:

- In `OnFetchAttestedCandidateRequest` when checking if the request bitfields have invalid size.
- In `handle_incoming_manifest_common` when the relay parent state cannot be obtained.

Additionally, there are TODO's [60] in the dispute coordinator code which also indicate missing reputation change handling.

Risk
The absence of reputation penalties in Kagome for invalid peer requests weakens the node's ability to isolate or deprioritize misbehaving peers, allowing them to repeatedly send invalid requests without consequences. This increases resource consumption (e.g., CPU, memory, and bandwidth) and can degrade the quality of peer connections, potentially disrupting statement distribution and candidate validation processes. Over time, the lack of consistent enforcement may lead to network inefficiencies and reduced resilience against malicious or poorly behaving nodes.

Mitigation recommendation
We recommend implementing the missing reputation penalty messages in Kagome for invalid requests and similar scenarios. Ensure that these penalties align with the Polkadot SDK's behavior to maintain consistency across implementations and enhance network resilience.

6.12 S2-2: Missing failure conditions in assignment certificate verification

Tracking	**[28]**
Severity	Medium
Status	Mitigated [29, 30, 31]

Issue description
The Polkadot SDK performs cryptographic verification on assignment certificates to validate a validator's claim (assignment) to approve a parachain candidate block. This ensures the claim is legitimate and adheres to protocol rules, like verifying cryptographic proofs and core indices. The Kagome implementation is missing some of these checks and verifications performed by Polkadot SDK.

Kagome's `checkAssingmentCert` [61] implementation lacks the following checks compared to Polkadot SDK [62]:

- `AssignmentKey` verification missing
- candidate claim enforcement check missing for `RelayVRFModulo` and `RelayVRFDelay` approvals
- `vrf_verify_extra` and `relay_vrf_modulo_core` checks are open TODOs [63].

Risk
Missing failure conditions in Kagome's assignment certificate verification reduce the accuracy of assignment and approval handling. By failing to detect invalid or improperly verified assignments, Kagome nodes risk producing incorrect approvals or failing to meet protocol expectations. In the Polkadot network, where most validators follow stricter verification rules, such errors are likely to trigger slashing mechanisms. Slashing penalties for invalid actions would lead to financial loss for Kagome-based validators, reputational damage, and reduced stake in the network, compromising their ability to participate effectively and weakening the overall security of the protocol.

Mitigation recommendation
We recommend implementing the missing checks in Kagome for assignment certificate verification. Ensure that these checks are aligned with the Polkadot SDK implementation to maintain consistency.

6.13 S1-1: Secure validator mode does not support clone instead of fork

Tracking	**[32]**
Severity	Low
Status	Mitigated [33]

Issue description
Kagome's Secure Validator Mode is missing support for spawning new job processes with clone instead of fork. This introduces a potential divergence in security and sandboxing capabilities between implementations. Compared to fork, clone allows to further isolate workloads through the configuration of additional sandboxing parameters, as can be seen in the Polkadot SDK pull request [64].

Risk
Using fork instead of clone risks inadequate isolation of subprocesses because fork doesn't support fine-grained control over namespaces. Subprocesses might share critical resources like the parent's network stack, IPC mechanisms, and filesystem mounts. This can expose the parent process to attacks such as privilege escalation or unauthorized access to shared resources, especially in environments with potentially adversarial workloads.

Mitigation recommendation
To align with Polkadot's implementation, we recommend that Kagome implements clone-based child process creation as part of the Secure Validator mode.

6.14 S1-2: Insecure input validation using debug_assert in unsafe functions

Tracking	**[34]**
Severity	Low
Status	Mitigated [35]

Issue description
The Rust crate `erasure-coding-crust`, used by Kagome, uses the `debug_assert` macro instead of the standard `assert` macro for input validation within functions marked as unsafe [65]. This practice relies on input assumptions that are not enforced in release builds. Debug assertions, by design, only execute in debug mode, leaving the validation checks entirely omitted in optimized release builds. Consequently, this exposes the potential for invalid inputs to bypass safety checks, leading to undefined behavior or application crashes during runtime. This issue becomes more critical in unsafe functions, where the caller is responsible for ensuring input validity, yet these functions lack enforced safeguards in production deployments.

Risk
The risk is the possibility of application crashes due to invalid inputs in release builds. While the occurrence is limited to misuse or invalid inputs in unsafe functions, the lack of runtime enforcement increases the likelihood of instability, especially in scenarios where input assumptions are inadvertently violated.

Mitigation recommendation
We recommend replacing `debug_assert` with standard `assert` statements for input validation in unsafe functions. This ensures that input assumptions are enforced consistently in both debug and release builds. The advantage of this approach is its simplicity and alignment with Rust's principles of safety and correctness. The disadvantage is the potential performance overhead introduced by runtime checks, which may be non-trivial in high-frequency code paths. Another option is to clearly document the use of `debug_assert` and provide comprehensive guidelines for users of the crate, emphasizing the need for thorough validation before invoking unsafe functions. This alternative preserves performance but requires rigorous diligence from developers to avoid misuse.

6.15 S0-1: Prospective parachains: missing test introduced with elastic scaling

Tracking	**[36]**
Severity	Info
Status	Mitigated [37, 38]

Issue description
Alongside the collator protocol changes for elastic scaling (validator side) pull request [66] in the Polkadot SDK, a new test [67] was added to verify behavior when incorrect parent head data is returned in a collation fetch response. The Kagome host implementation currently lacks this test.

Risk
Missing tests in Kagome that are implemented in the Polkadot SDK pose a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation.

Mitigation recommendation
To address this issue, we recommend integrating existing tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

Furthermore, to make checking for conformance easier, we recommend naming the Kagome tests consistently with the Polkadot SDK tests or adding links to the corresponding tests.

6.16 S0-2: Missing support for runtime API function candidates_pending_availability

Tracking	**[39]**
Severity	Info
Status	Mitigated [40]

Issue description
For elastic scaling, the runtime API was adjusted [68] to include a new `candidates_pending_availability` function, deprecating the `candidate_pending_availability` function. The Kagome host implementation currently has no support for this new function.

Risk
When the deprecated candidate_pending_availability function is removed, all calls to it in the Kagome host implementation will fail, potentially disrupting node functionality.

Mitigation recommendation
We recommend adding support for the `candidates_pending_availability` function and adjusting all usages of the deprecated `candidate_pending_availability` function to ensure future compatibility with the runtime API.

6.17 S0-3: FragmentTree_checkBackableQueryMultipleCandidates missing testcases

Tracking	**[41]**
Severity	Info
Status	Mitigated [42]

Issue description
A follow up pull request [69] to prospective parachains in the Polkadot sdk introduced some additional tests for prospective parachains which are missing in kagome.

Risk
Missing tests in Kagome that are implemented in the Polkadot SDK pose a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation.

Mitigation recommendation
To address this issue, we recommend integrating existing tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

6.18 S0-4: Prospective parachains: backing tests missing

Tracking	**[41]**
Severity	Info
Status	Mitigated [44, 45]

Issue description
The Polkadot SDK includes multiple tests [70] for the backing subsystem with prospective parachains enabled, which are absent in Kagome. Specifically, the missing tests are:

- `seconding_sanity_check_allowed_on_all`
- `seconding_sanity_check_disallowed`
- `seconding_sanity_check_allowed_on_at_least_one_leaf`
- `prospective_parachains_reject_candidate`
- `second_multiple_candidates_per_relay_parent`
- `backing_works`
- `concurrent_dependent_candidates`
- `seconding_sanity_check_occupy_same_depth`
- `occupied_core_assignment`

Risk
Missing tests in Kagome that are implemented in the Polkadot SDK pose a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation.

Mitigation recommendation
To address this issue, we recommend integrating existing tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

6.19 S0-5: Grid topology: missing tests

Tracking	**[46]**
Severity	Info
Status	Mitigated

Issue description
The Polkadot SDK implements extensive tests for the grid topology in two different files ([71], [72]). To the best of our knowledge, all these tests are missing in Kagome. There are some grid tests implemented [73]; however, they are not very extensive and are hard to follow due to the use of hardcoded values.

Risk
Missing tests in Kagome that are implemented in the Polkadot SDK pose a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation.

Mitigation recommendation
To address this issue, we recommend integrating existing tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

6.20 S0-6: Missing tests for DHT authority discovery record creation time

Tracking	**[47]**
Severity	Info
Status	Mitigated [48]

Issue description

The Polkadot SDK implements extensive tests [74] for the DHT Authority discovery record creation time feature. All these tests are absent in Kagome. Specifically, the missing tests are:

- `test_quickly_connect_to_authorities_that_changed_address`
- `strict_accept_address_without_creation_time`
- `keep_last_received_if_no_creation_time`
- `records_with_incorrectly_signed_creation_time_are_ignored`
- `newer_records_overwrite_older_ones`
- `older_records_dont_affect_newer_ones`

Risk

Missing tests in Kagome that are implemented in the Polkadot SDK pose a risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation.

Mitigation recommendation

To address this issue, we recommend integrating existing tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

6.21 S0-7: Systematic chunks recovery: missing tests

Tracking	**[49]**
Severity	Info
Status	Mitigated [50]

Issue description
Polkadot SDK includes multiple tests [75] for the availability recovery from systematic chunks. The Kagome host implementation also includes tests [76] for systematic chunks; however, these tests are not as exhaustive as those in the Polkadot SDK.

Risk
Missing tests in Kagome that are implemented in the Polkadot SDK pose a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation.

Mitigation recommendation
To address this issue, we recommend integrating existing tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

6.22 S0-8: Lack of parachain tests Leads to unverified implementation consistency

Tracking	**[51]**
Severity	Info
Status	Acknowledged

Issue description
Kagome currently only implements Zombienet tests, omitting any parachain tests from Polkadot. This exclusion raises concerns regarding the verification of its implementation's consistency and compatibility with Polkadot's ecosystem. Parachain tests are crucial for ensuring that implementations like Kagome can integrate seamlessly with Polkadot's parachains, which are essential for the network's interoperability and scalability.

Risk
The absence of parachain tests in Kagome's testing framework poses a high risk of undetected discrepancies between Kagome and Polkadot's expected behavior, potentially leading to integration issues, security vulnerabilities, and decreased trust in Kagome's reliability as a Polkadot Runtime Environment implementation. This gap undermines the confidence in Kagome's ability to support the Polkadot ecosystem's multi-chain architecture effectively.

Mitigation recommendation
To address this issue, we recommend integrating existing parachain tests from Polkadot into Kagome's Testing Framework. Directly leveraging Polkadot's existing parachain tests ensures that Kagome is tested against the same criteria as the Polkadot network, promoting consistency and interoperability. This approach would likely be the most straightforward way to validate Kagome's compatibility with Polkadot.

7 Bibliography

[1] [Online]. Available: https://github.com/qdrvm/kagome/tree/v0.9.5.

[2] [Online]. Available: https://clang.llvm.org/docs/ClangStaticAnalyzer.html.

[3] [Online]. Available: https://cppcheck.sourceforge.io/.

[4] [Online]. Available: https://semgrep.dev/.

[5] [Online]. Available: https://github.com/qdrvm/KAGOME-audit.

[6] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/37.

[7] [Online]. Available: https://github.com/qdrvm/kagome/pull/2231.

[8] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/36.

[9] [Online]. Available: https://github.com/qdrvm/kagome/pull/2240.

[10] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/52.

[11] [Online]. Available: https://github.com/qdrvm/scale-codec-cpp/pull/28.

[12] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/35.

[13] [Online]. Available: https://github.com/qdrvm/kagome/pull/2231.

[14] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/41.

[15] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/44.

[16] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/34.

[17] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/30.

[18] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/31.

[19] [Online]. Available: https://github.com/qdrvm/kagome/pull/1979.

[20] [Online]. Available: https://github.com/qdrvm/kagome/pull/1992.

[21] [Online]. Available: https://github.com/qdrvm/kagome/pull/2091.

[22] [Online]. Available: https://github.com/qdrvm/kagome/pull/2438.

[23] [Online]. Available: https://github.com/qdrvm/kagome/pull/2411.

[24] [Online]. Available: https://github.com/qdrvm/qtils/pull/16.

[25] [Online]. Available: https://github.com/qdrvm/kagome/pull/2435.

[26] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/45.

[27] [Online]. Available: https://github.com/qdrvm/kagome/pull/2407.

[28] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/48.

[29] [Online]. Available: https://github.com/qdrvm/kagome/pull/2389.

[30] [Online]. Available: https://github.com/qdrvm/kagome-crates/pull/9.

[31] [Online]. Available: https://github.com/qdrvm/kagome/pull/2412.

[32] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/38.

[33] [Online]. Available: https://github.com/qdrvm/kagome/pull/2307.

[34] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/49.

[35] [Online]. Available: https://github.com/qdrvm/erasure-coding-crust/pull/12.

[36] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/40.

[37] [Online]. Available: https://github.com/qdrvm/kagome/pull/2270.

[38] [Online]. Available: https://github.com/qdrvm/kagome/pull/2426.

[39] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/42.

[40] [Online]. Available: https://github.com/qdrvm/kagome/pull/2253.

[41] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/43.

[42] [Online]. Available: https://github.com/qdrvm/kagome/pull/2428.

[43] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/46.

[44] [Online]. Available: https://github.com/qdrvm/kagome/pull/2315.

[45] [Online]. Available: https://github.com/qdrvm/kagome/pull/2413.

[46] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/47.

[47] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/50.

[48] [Online]. Available: https://github.com/qdrvm/kagome/pull/2311.

[49] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/51.

[50] [Online]. Available: https://github.com/qdrvm/kagome/pull/2321.

[51] [Online]. Available: https://github.com/qdrvm/KAGOME-audit/issues/32.

[52] [Online]. Available:](https://github.com/paritytech/polkadot-sdk/blob/f6d08e637694729df7cddb166954ee90c2da3ce1/polkadot/node/core/dispute-coordinator/src/lib.rs#L493.

[53] [Online]. Available: https://github.com/qdrvm/kagome/blob/7b71057c550a7bcde27d5d0b143f41f8360b251d/core/dispute_coordinator/impl/dispute_coordinator_impl.cpp#L1362.

[54] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/e8dad101634748a164a26ed8f404f218af29789e/polkadot/node/core/dispute-coordinator/src/tests.rs#L2756.

[55] [Online]. Available: https://github.com/qdrvm/kagome/blob/7b71057c550a7bcde27d5d0b143f41f8360b251d/core/dispute_coordinator/impl/dispute_coordinator_impl.cpp#L1406.

[56] [Online]. Available: https://paritytech.github.io/polkadot-sdk/book/node/disputes/dispute-coordinator.html#disabling.

[57] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/f6d08e637694729df7cddb166954ee90c2da3ce1/polkadot/node/core/dispute-coordinator/src/initialized.rs#L98.

[58] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/c422d8bbae8ba327597582203f05d30c26ef1392/polkadot/node/network/statement-distribution/src/v2/mod.rs#L3332.

[59] [Online]. Available: https://github.com/qdrvm/kagome/blob/2433e357341bed6fc2207f544395cc80e7c9550d/core/parachain/validator/impl/parachain_processor.cpp#L3222.

[60] [Online]. Available: https://github.com/qdrvm/kagome/blob/2433e357341bed6fc2207f544395cc80e7c9550d/core/dispute_coordinator/impl/dispute_coordinator_impl.cpp#L2198.

[61] [Online]. Available: https://github.com/qdrvm/kagome/blob/5418b4e91baffac42ec0532d4536dd8da9d60b54/core/parachain/approval/approval_distribution.cpp#L456.

[62] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/f7838db506f48e48671f867f23d8c12858c5b67c/polkadot/node/core/approval-voting/src/criteria.rs#L539.

[63] [Online]. Available: https://github.com/qdrvm/kagome/blob/5418b4e91baffac42ec0532d4536dd8da9d60b54/core/parachain/approval/approval_distribution.cpp#L521C31-L521C47.

[64] [Online]. Available: https://github.com/paritytech/polkadot-sdk/pull/2477.

[65] [Online]. Available: https://github.com/qdrvm/erasure-coding-crust/blob/4eb9953b4b9aa5199e015d3774cb8631fb4872da/src/erasure_coding.rs#L349.

[66] [Online]. Available: https://github.com/paritytech/polkadot-sdk/pull/3302.

[67] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/5d7181cda209053fda007679e34afccfa3e95130/polkadot/node/network/collator-protocol/src/validator_side/tests/prospective_parachains.rs#L822.

[68] [Online]. Available: https://github.com/paritytech/polkadot-sdk/pull/4027.

[69] [Online]. Available: https://github.com/paritytech/polkadot-sdk/pull/3233.

[70] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/e98c1ac6994766411f96bd7c14e8049cc5284396/polkadot/node/core/backing/src/tests/prospective_parachains.rs#L1.

[71] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/9ec8009c5a8fdf89499fcd2a40df0292d3950efa/polkadot/node/network/statement-distribution/src/v2/grid.rs#L1095.

[72] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/9ec8009c5a8fdf89499fcd2a40df0292d3950efa/polkadot/node/network/statement-distribution/src/v2/tests/grid.rs#L25.

[73] [Online]. Available: https://github.com/qdrvm/kagome/blob/v0.9.5/test/core/parachain/grid.cpp.

[74] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/e1add3e8a8faa611e63e3f962b6c5b13ba37e449/substrate/client/authority-discovery/src/worker/tests.rs#L1.

[75] [Online]. Available: https://github.com/paritytech/polkadot-sdk/blob/master/polkadot/node/network/availability-recovery/src/tests.rs.

[76] [Online]. Available: https://github.com/qdrvm/kagome/blob/v0.9.5/test/core/parachain/availability/recovery_test.cpp.

[77] [Online]. Available: https://github.com/srlabs/ziggy.

[78] [Online]. Available: https://a.b.

[79] [Online]. Available: https://github.com/client/git.

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

Appendix A: Security Research Labs technical services

Security Research Labs delivers extensive technical expertise to meet your security needs. Our comprehensive services include software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. We aim to equip your organization with the security knowledge essential for achieving your objectives.

SOFTWARE EVALUATION We provide assessments of applications, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of applications. Our work includes design and architecture reviews, data flow and threat modelling, and code analysis with targeted fuzzing to find exploitable issues.

BLOCKCHAIN SECURITY ASSESSMENTS We offer specialized security assessments for blockchain technologies, focusing on the unique challenges posed by decentralized systems. Our services include smart contract audits, consensus mechanism evaluations, and vulnerability assessments specific to blockchain infrastructure. Leveraging our deep understanding of blockchain technology, we ensure your decentralized applications and networks are secure and robust.

POLKADOT ECOSYSTEM SECURITY We provide comprehensive security services tailored to the Polkadot ecosystem, including parachains, relay chains, and cross-chain communication protocols. Our expertise covers runtime misconfiguration detection, benchmarking validation, cryptographic implementation reviews, and XCM exploitation prevention. Our goal is to help you maintain a secure and resilient Polkadot environment, safeguarding your network against potential threats.

TELCO SECURITY We deliver specialized security assessments for telecommunications networks, addressing the unique challenges of securing large-scale and critical communication infrastructures. Our services encompass vulnerability assessments, secure network architecture reviews, and protocol analysis. With a deep understanding of telco environments, we ensure robust protection against cyber threats, helping maintain the integrity and availability of your telecommunications services.

DEVICE TESTING Our comprehensive device testing services cover a wide range of hardware, from IoT devices and embedded systems to consumer electronics and industrial controls. We perform rigorous security evaluations, including firmware analysis, penetration testing, and hardware-level assessments, to identify vulnerabilities and ensure your devices meet the highest security standards. Our goal is to safeguard your hardware against potential attacks and operational failures.

CODE AUDITING We provide in-depth code auditing services to identify and mitigate security vulnerabilities within your software. Our approach includes thorough manual reviews, automated static analysis, and targeted fuzzing to uncover critical issues such as logic flaws, insecure coding practices, and exploitable vulnerabilities. By leveraging our expertise in secure software development, we help you enhance the security and reliability of your codebase, ensuring robust protection against potential threats.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated attackers. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team a sense of the overall security posture of your organization.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology, allowing us to find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This gives your team a stronger assurance that the significant security-impacting flaws have been found and corrected.

SECURITY DEVELOPMENT LIFECYCLE CONSULTING We guide organizations through the Security Development Lifecycle to integrate security at every phase of software development. Our services include secure coding training, threat modelling, security design reviews, and automated security testing implementation. By embedding security practices into your development processes, we help you proactively identify and mitigate vulnerabilities, ensuring robust and secure software delivery from inception to deployment.

REVERSE ENGINEERING We assist clients with reverse engineering efforts not associated with malware or incident response. We also provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, to embedded systems, to mobile devices, to consumer-facing end products, to core networking equipment that powers Internet backbones.

VULNERABILITY PRIORITIZATION We streamline vulnerability information processing by consolidating data from compliance checks, audit findings, penetration tests, and red team insights. Our prioritization and automation strategies ensure that the most critical vulnerabilities are addressed promptly, enhancing your organization's security posture. By systematically categorizing and prioritizing risks, we help you focus on the most impactful threats, ensuring efficient and effective remediation efforts.

SECURITY MATURITY REVIEW We conduct comprehensive security maturity reviews to evaluate your organization's current security practices and identify areas for improvement. Our assessments cover a wide range of criteria, including policy development, risk management, incident response, and security awareness. By benchmarking against industry standards and best practices, we provide actionable insights and recommendations to enhance your overall security posture and guide your organization toward achieving higher levels of security maturity.

SECURITY TEAM INCUBATION We provide comprehensive support for building security teams for new, large-scale IT ventures. From Day 1, our ramp-up program offers essential security advisory and assurance, helping you establish a robust security foundation. With our proven track record in securing billion-dollar investments and launching secure telco networks globally, we ensure your new enterprise is protected against cyber threats from the start.

HACKING INCIDENT SUPPORT We offer immediate and comprehensive support in the event of a hacking incident, providing expert analysis, containment, and remediation. Our services include detailed forensics, malware analysis, and root cause determination, along with actionable recommendations to prevent future incidents. With our rapid response and deep expertise, we help you mitigate damage, recover swiftly, and strengthen your defenses against potential threats.