Joystream Baseline Security Assurance

Threat model and hacking assessment report

V0.9, March 3, 2021

Regina Bíró	regina@srlabs.de
Stephan Zeisberg	stephan@srlabs.de
Mostafa Sattari	mostafa@srlabs.de
Vincent Ulitzsch	vincent@srlabs.de

Abstract. This study describes the results of a thorough, independent baseline security assurance audit of the Joystream blockchain platform performed by Security Research Labs. In the course of this study, Jsgenesis provided full access to relevant documentation and supported the research team effectively.

The protection of Joystream was independently verified to assure that existing hacking risks are understood and minimized.

The research team identified several issues ranging from low to critical risk and Jsgenesis started addressing them in a timely manner.

To further improve the security of the Joystream network, we recommend introducing deposits for on-chain storage to avoid DoS attacks by storage cluttering. We also strongly support conducting an audit on the economic incentives; as well as establishing a secure-by-design approach by adhering to best practices as detailed in this document.

Content

1 Motivation and scope

This review assesses the Joystream blockchain system's existing protections against a variety of **likely hacking scenarios** and **points out the most relevant weaknesses**, all with the goal of improving the protection capabilities of the blockchain system.

Threats that could compromise the Joystream network go far beyond the theft of tokens. Notable hacking scenarios include the potential to undermine trust in the blockchain system by member account takeover, influencing community decisions or bypassing staking requirements.

This report details the baseline security assurance results with the aim of creating transparency in three steps:

Threat Model. 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 nodes in future Joystream systems. For each hacking incentive, *hacking scenarios* were postulated*,* by which these goals could be reached. The threat model provides guidance for the design, implementation, and security testing of Joystream.

Security design coverage check. Next, the Joystream design was reviewed for coverage against relevant hacking scenarios. For each scenario, the following two aspects were investigated:

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

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

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

Joystream is built upon Substrate, a blockchain development framework. Both Joystream and Substrate are written in Rust, a memory safe programming language. Mainly, Substrate works with three technologies: a WebAssembly (WASM) based runtime, decentralized communication via libp2p, GRANDPA finality gadget and the BABE block production engine.

The Joystream 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.

Jsgenesis shared an overview containing the current state of the runtime modules used by Joystream and its audit priority. The in-scope components and the assigned priorities are reflected in Table 1. SRLabs conducted the audit fully focusing on the components indicated as in scope by Jsgenesis. The documentation in Joystream's handbook [1] provided the testers a good runtime module design and implementation overview.

Repository	Priority	Component(s)
Joystream	High	Membership, Proposals, Staking handler, Working group
	Medium	Node, Chain-spec builder, Runtime, Forum, Common, Constitution, Council, Referendum

Table 1. In-scope Joystream components with audit priority

2 Methodology

To be able to effectively review the Joystream codebase, a threat-model driven code review strategy was employed. For each identified threat, hypothetical attacks that can be used to realize the threat were developed and mapped to their respective threat category as outlined in chapter 3.

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 pallets.
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, ensure 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.

During the audit, we carried out a hybrid strategy utilizing a combination of code review and dynamic tests (e.g. fuzz-testing) to assess the security of the Joystream codebase.

While fuzz-testing and dynamic tests establish a baseline assurance, the main focus of this audit was a manual code review of the Joystream codebase to identify logic bugs, design flaws, and best practice deviations. We used the *olympia* branch (be26ea167) of the Joystream repository as the basis for the review. 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 Joystream 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 Joystream's case is mostly the functions implementing the extrinsics. (Note that the network part is handled by Substrate, which was not in scope for this review, but is built with a strong emphasis on security and where fuzz-testing is also used). 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.

During the course of the audit, findings were shared via a private Github repository [2]. This repository was also utilized to document status updates and the overall review progress – in addition, weekly jour fixe meetings were held to provide detailed updates and address open questions.

3 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in Joystream'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*, takes into account 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 preforming 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 somewhat 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.

A more detailed description of the threat modelling framework is summarized in the shared Github repository [3].

After applying the framework to the Joystream system, different threat scenarios according to the CIA triad (Confidentiality, Integrity and Availability) were identified. Table 2 provides a high-level overview of the threat model 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 [4].

Security promise	**Hacking value**	**Example threat scenarios**	**Hacking effort**	**Example attack ideas**
Confiden-tiality	Medium	- Revealing sealed votes of other members	High	- Calculating hash preimage of sealed votes
Integrity	High	- Taking over a working group - Manipulating the slashing mechanism - Gaining native tokens for free - Influencing proposal discussions	Low	- Gaining free tokens by abusing the invite system - Exploiting a logic bug to manipulate slashing mechanism - Spoofing a member's ID in discussions
Availability	High	- Blocking all the proposal slots - Harming chain functionality by spamming/DoSing	Low	- Filling up blockchain storage by calling extrinsics that are missing storage deposits - Crafting heavy-weight extrinsics to block others from making it into blocks - Spamming discussion threads up to their limits

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

4 Findings summary

We identified 14 issues - summarized in Table 3 - during our analysis of the runtime modules in scope in the Joystream codebase that enable the attacks outlined above. In addition, we also reported some minor bugs and best practice deviations. In summary, 1 critical severity, 2 high severity, 4 medium severity issues, 5 low severity and 2 information level issues were found.

Issue description	Severity	Reference	Remediation
Exponential complexity of weight calculation functions in multiple imported Substrate v2 pallets	Critical	[5]	In progress
The existential deposit is configured to be 0	High	[6]	PR 2187 [7]
No deposit is charged for adding staking account candidates to a membership	High	[8]	In progress
No deposit is charged for creating forum posts and threads	Moderate	[9]	In progress
The *set_stickied_threads* extrinsic can be used to cheaply fill up the blockchain storage	Moderate	[10]	PR 2187 [7]
Account takeover via malicious member invitation	Moderate	[11]	In progress
No deposit is charged for applying on working group openings	Moderate	[12]	In progress
Members can undermine the intent of the *invite_member* extrinsic by inviting themselves to get free tokens	Low	[13]	PR 2187 [7]
In referendums, the voting stake is only locked for balances transfer	Low	[14]	PR 2187 [7]
Multiple votes per user possible in the forum pallet via *vote_on_poll*	Low	[15]	PR 2187 [7]
No deposit is charged for adding position openings for working groups	Low	[16]	In progress
No deposit is charged for adding posts on proposal threads	Low	[17]	In progress
Missing checks on the *referral_cut* value could enable creating unlimited membership accounts for free	Info	[18]	PR 2187 [7]
In the commit-reveal voting scheme for referendums, UUID4 is used for random salt generation	Info	[19]	In progress

Table 3. Overview of identified issues

4.1 Detailed findings

4.1.1 Exponential complexity of weight calculation functions allows for DoS

The weight calculation functions of the *batch, as_derivative* (in frame/utility/src/lib.rs), *sudo* and *sudo_as* (in frame/sudo/src/lib.rs) extrinsics have exponential complexity because they perform two calls to the *get_dispatch_info()* function. An attacker can abuse the exponential complexity of *O(2^n)* to craft a nested extrinsic (that is still below the *MAX_EXTRINSIC_DEPTH* of 256), for which the weight computation is not feasible in limited time and thus cause a validator to miss its slot and fail at block production, potentially halting block production.

The same problem existed in other extrinsics in Substrate as well. To mitigate exponential complexity in other extrinsics, we suggest updating the Substrate version Joystream is using to a version that includes PR 7849 [20], which fixes the respective problems in Substrate. Joystream is planning to update to Substrate v3 [21] which automatically mitigates the issue.

4.1.2 No deposit for parameters stored on-chain can lead to storage clutter

Throughout several pallets in the Joystream codebase, deposits are not charged in accordance with the storage used when certain data is written and stored on chain. This could allow attackers to cheaply fill up storage by calling extrinsics not accounting for storage ([8], [9], [10], [12], [16], [17]) or creating a high number of accounts and distributing some native tokens among them ([5]).

While transaction fees reduce the risks of these attacks, they do not fully mitigate them – we recommend charging deposits for on-chain storage and setting the existential deposit to a value greater than 0. Joystream already mitigated the missing existential deposit in PR 2187 [7] and is planning to add the logic for requiring deposits for data stored on chain in the future.

4.1.3 Account takeover via malicious invitation

In the membership invitation system for Joystream, inviting members are free to choose the *root_account* and *controller_account* for the invitee. Setting a *root_account* that is controlled by the malicious inviting member could result in an account takeover for the new member at a later point in time [11].

To mitigate this issue, we suggest that Joystream raises user awareness of the security implications of a compromised *root_account* during membership invitations, for example via updating the documentation on membership invitations.

4.1.4 Minor issues

During our analysis we discovered several minor bugs in the Joystream runtime modules. While these vulnerabilities are of low or info-level severity, we recommend implementing fixes to them in order to avoid attackers combining these vulnerabilities with other undiscovered or later introduced bugs for a higher impact.

Membership: free tokens via membership invitations [13]**.** An attacker could transfer the *invitation_balance* set to one's own account, by inviting one's own account id. It was also possible to invite an *AccountId* which already had a membership, which undermined the intent of the invitation system. Joystream mitigated this issue in PR 2187 [7] by adding the logic for checking whether an invitation lock is already set for the inviting account.

Membership: free membership via missing checks on the *referral_cut* value [18]**.** In the membership pallet, a *referral_cut* value can be configured that determines a referral bonus to incentivize inviting new members for existing members. The special case when the *referral_cut* == *membership_fee* enabled any user to create unlimited new membership accounts for free. Joystream mitigated this issue in PR 2187 [7] by changing the *referral_cut* to be maximum 50% of the *membership_fee*.

Referendum: the voting stake is only locked for balances transfer [14]**.** In the referendum pallet, in *fn vote* the voting stake was only locked for balances transfer, but not for paying fees, tips or transaction costs; enabling users to use up their staked balance for these actions. Joystream mitigated this issue in PR 2187 [7] by changing the limits on the staking lock to *WithdrawReasons::all()*.

Forum: multiple votes per user on forum polls [15]**.** In the forum pallet, the *vote_on_poll* extrinsic did not have a check implemented on whether a user has already voted for a specific poll. Joystream mitigated this issue in PR 2187 [7] by adding the logic to check whether a user have already voted on a poll.

5 Evolution suggestion

Parts of the process towards a security-mature product are conducting thorough, regular reviews of the Joystream's critical codebase that help to harden the codebase and train developers to cultivate a security mindset.

Moving forward, we suggest taking the following measures to harden Joystream's codebase against potential security vulnerabilities introduced in future development.

Introduce deposits for on-chain storage usage. A significant proportion of the identified issues come from missing deposits for on-chain data storage. To prevent denial of service attacks against the blockchain network, we strongly recommend requiring a deposit for data that is written to the blockchain storage. The Polkadot and Kusama runtime implementations are good examples on how to configure base and per-byte deposit fees.

Audit on economic incentives. While – with the exception of the existential deposit value – we did not identify direct security issues in the token and price configuration values (e.g. byte-length transaction fee, deposits, etc), it was out of scope of our analysis whether they support a sound economic model that is aligned with Joystream's objectives. Therefore, we suggest conducting an independent economic audit on the Joystream system before the main-net release.

Regular updates. According to our knowledge, Parity Technologies does not provide security advisories for Substrate yet; therefore, new releases of Substrate may contain fixes of critical security issues. Since Joystream is a product that heavily relies on Substrate, we recommend updating to the latest version as soon as possible, when there is a new available release.

Regular code review and continuous fuzz-testing. While we had the overall impression that the Joystream code is well written and is free of arithmetic bugs, underweighted extrinsics, reachable assertions or unbound allocations; it is important to keep regularly reviewing the codebase (for example, internally) to avoid introducing new logic or arithmetic bugs. We also recommend employing fuzz-testing to identify potential vulnerabilities early on in the development process. Ideally, Jsgenesis would continuously fuzz their code on each commit made to the codebase. Jsgenesis can draw some inspiration from the Substrate codebase, which includes multiple fuzzing harnesses based on honggfuzz [22].

Use saturating arithmetic wherever possible. During the audit, no security issues were found that stem from using unsafe arithmetic operations that could under/overflow. During the manual analysis, we found that saturating arithmetic was used throughout the pallets in scope – we suggest keeping this best practice for the implementation of future features as well.

Expand unit/functional testing. Unit testing can ensure that all code meets not only quality, but also security standards before its deployed. Security unit tests can for example help ensure that critical operations are only permitted to authorized actors. We recommend writing unit tests for as many identified threats as possible. Functional testing could uncover undocumented misuses and documentation discrepancies that could undermine the security of the network before releasing a new version. Functional tests conducted from a user's perspective can help highlight gaps in the documentation and users's understanding of security-critical operations (for example compromising a *root_account*) and help developers uncover by-default permitted actions that users could use to abuse the system.

Leverage internal threat modeling to spread awareness between developers. Since developers have a deep understanding of the codebase, identifying and prioritizing potential threats as part the development process would enhance the security of the Joystream codebase.

6 References

[1] "Joystream Handbook," [Online]. Available: https://joystream.gitbook.io/joystream-handbook/.

[2] [Online]. Available: https://github.com/Joystream/audits.

[3] [Online]. Available: https://github.com/Joystream/audits/tree/main/SRL-Jsgenesis_baseline_security_assurance_joystream-2021/0-deliverables.

[4] "Joystream Threat Model," [Online]. Available: https://securityresearchlabs.sharepoint.com/:x:/s/Joystream/Ed53-ZkfyF5PrS1PHi9JhmAB3jbk4373Mk-0au0GPmdrkQ?e=80lCdG.

[5] [Online]. Available: https://github.com/Joystream/audits/issues/1.

[6] [Online]. Available: https://github.com/Joystream/audits/issues/9.

[7] [Online]. Available: https://github.com/Joystream/joystream/pull/2187.

[8] [Online]. Available: https://github.com/Joystream/audits/issues/11.

[9] [Online]. Available: https://github.com/Joystream/audits/issues/5.

[10] [Online]. Available: https://github.com/Joystream/audits/issues/6.

[11] [Online]. Available: https://github.com/Joystream/audits/issues/7.

[12] [Online]. Available: https://github.com/Joystream/audits/issues/14.

[13] [Online]. Available: https://github.com/Joystream/audits/issues/3.

[14] [Online]. Available: https://github.com/Joystream/audits/issues/8.

[15] [Online]. Available: https://github.com/Joystream/audits/issues/10.

[16] [Online]. Available: https://github.com/Joystream/audits/issues/12.

[17] [Online]. Available: https://github.com/Joystream/audits/issues/13.

[18] [Online]. Available: https://github.com/Joystream/audits/issues/2.

[19] [Online]. Available: https://github.com/Joystream/audits/issues/4.

[20] [Online]. Available: https://github.com/paritytech/substrate/pull/7849.

[21] [Online]. Available: https://github.com/paritytech/substrate/releases/tag/v3.0.0.

[22] [Online]. Available: https://github.com/google/honggfuzz.

[23] [Online]. Available: https://github.com/Joystream/audits/issues/11.