Joystream Security Audit

Audit of pallets and configuration

Reference	22-05-982-REP
Version	1.1
Date	2022/05/31

Quarkslab SAS
13 rue Saint Ambroise
75011 Paris
France

Contents

1 Project Information

Document history			
Version	**Date**	**Details**	**Authors**
1.1	2022/05/31	Initial Version	Christian Heitman and Quarkslab auditor.[1]

Quarkslab		
Contact	**Role**	**Contact Address**
Frédéric Raynal	CEO	`fraynal@quarkslab.com`
Stavia Salomon	Sales Manager	`ssalomon@quarkslab.com`
Matthieu Ramtine Tofighi Shirazi	Project Manager	`mrtofighishirazi@quarkslab.com`
Christian Heitman	R&D Engineer	`cheitman@quarkslab.com`
Quarkslab auditor[1]	R&D Engineer	
Mahé Tardy	R&D Engineer	`mtardy@quarkslab.com`

Joystream		
Contact	**Role**	**Contact Address**
Bedeho Mender	CEO	`bedeho@jsgenesis.com`
Mokhtar Namaani	CTO	`mokhtar@jsgenesis.com`
Shamil Gadelshin	Blockchain Engineer	`shamil@jsgenesis.com`

[1] Given the public release of this report, some auditors preferred to remain anonymous.

2 Executive Summary

This report describes the results of the security evaluation made by Quarkslab on multiple components developed by Joystream.

The main components reviewed were the `storage` pallet, the `content` pallet, and the `bounty` pallet. Their configuration and integration into the runtime were also investigated.

Audits have already been performed on some components by another audit company. During Quarkslab's assessment, one vulnerability, which is not exploitable with the current configuration of the runtime, and a few functional issues have been found. The most important issues were reported to the Joystream team during the audit and are either fixed, or being actively worked on, at the time this report has been written.

The report is composed of details and many recommendations on flaws that were discovered during the audit.

2.1 Disclaimer

This report reflects the work and results obtained within the duration of the audit on the specified scope (see. Section 3.2) as agreed between Joystream and Quarkslab. Tests are not guaranteed to be exhaustive and the report does not ensure the code to be bug-free.

2.2 Findings Summary

The severity classification, reflects a relative hierarchy between the various findings of this report. For example, the only high rating vulnerability is not exploitable in practice, but in order to reflect its importance and the possiblity that it becomes easily exploitable, it was promoted to high severity. The following table describes in more details the rating for findings severity levels.

Severity	Description
`Critical`	Exploitable major issues that could result in loss of funds or DDoS attack.
`High`	Exploitable major issues that could result in loss of funds or DDoS attack and whose setup has high requirements. Also issues that are likely to become exploitable and whose exploitation would be trivial.
`Medium`	Medium issues that cannot be directly exploited, such as the use of unsafe arithmetic or potential panics. These issues could potentially lead to loss of funds or DDoS attacks in future updates. Also non-security-related issues that have a high impact on the working of the chain.
`Low`	Low issues that cannot be directly exploited such as mismatch between the specification and implementation on pre/post conditions or incorrect weight. These issues could potentially lead to logic bugs and cheap computation or storage.

Severity	Description
Info	Diverse informative recommendations on code structure, documentation, TODO annotations, etc.

ID	Description	Category	Severity
HIGH_1	A logic bug could allow to steal funds from the Treasury	Logic bug	High
MEDIUM_1	Update the Substrate version	Outdated dependencies	Medium
MEDIUM_2	The `deletion_prize` parameter is not bounded	Missing precondition, partial Denial of service	Medium
MEDIUM_3	The `deletion_prize` parameter is not bounded	Missing precondition, partial Denial of service	Medium
MEDIUM_4	The `new_data_size_fee` parameter is not bounded	Missing precondition, partial Denial of service	Medium
MEDIUM_5	Usage of unsafe addition	Unsafe arithmetic, partial Denial of service	Medium
MEDIUM_6	The `issue_nft` extrinsic fails on open auctions	Functionality not working	Medium
MEDIUM_7	The payment for a non-fungible token can go to the wrong user	Wrong postcondition, fund loss	Medium
MEDIUM_8	Funds can stay locked forever	Missing postcondition, fund loss	Medium
MEDIUM_9	A logic bug that could allow previous channel collaborators manipulate its assets	Logic bug	Medium
MEDIUM_10	Funds of participants can stay locked	Missing precondition/postcondition	Medium
LOW_1	Usage of unsafe addition	Unsafe arithmetic	Low
LOW_2	Fixed parameter in the benchmark	Benchmark	Low
LOW_3	No check on `transactor_account_id`	Missing precondition	Low
LOW_4	`validate_update_distribution_bucket` is missing some corner cases	Logic bug	Low
LOW_5	`distribution_bucket_id` can be optimized	Optimization	Low
LOW_6	Unsafe multiplication in `compute_net_prize`	Unsafe arithmetic	Low
LOW_7	All the extrinsics have a default weight value	Missing benchmark, `TODO` annotation	Low
LOW_8	The extrinsic can fail because of a race condition	Race condition	Low

ID	Description	Category	Severity
LOW_9	The `to` parameter is not checked	Missing precondition	Low
LOW_10	Use of `can_slash`	Postcondition can be false	Low
LOW_11	The `end` parameter is not checked	Missing precondition	Low
LOW_12	Top bidder requires more funds to bid again	Logic bug	Low
LOW_13	Check in `ensure_constraints_on_bid_amount` should be $>=$	Logic bug	Low
LOW_14	Missing boundary check	Boundary check	Low
LOW_15	Missing boundary check	Boundary check	Low
LOW_16	Multiple extrinsics can fail because of race conditions	Race condition	Low
LOW_17	Possible inconsistent check of the `NoActiveTransfer` flag across extrinsics	Inconsistent usage	Low
LOW_18	Possible missing parameter when transferring a channel	Missing parameter	Low
LOW_19	No upper bound for `funding_period`	Missing precondition	Low
LOW_20	Unsafe arithmetic in `funding_period_expired`	Unsafe arithmetic	Low
LOW_21	Unsafe increment of the `entry_count`	Unsafe arithmetic	Low
LOW_22	Missing event in case of reject	Missing postcondition	Low
INFO_1	Use the transactional macro	Error handling	Info
INFO_2	The `update_data_object_deletion_priz` extrinsic is not benchmarked	Missing benchmark, `TODO` annotation	Info
INFO_3	The `update_dynamic_bag_deletion_prize` extrinsic is not benchmarked	Missing benchmark, `TODO` annotation	Info
INFO_4	Unsafe increment in `increment_distribution_family_numbe`	Unsafe arithmetic	Info
INFO_5	Unsafe increment in `increment_next_distribution_bucket_index_counter`	Unsafe arithmetic	Info
INFO_6	The `sudo_upload_data_objects` extrinsic is not benchmarked	Missing benchmark, `TODO` annotation	Info
INFO_7	The `sudo_create_dynamic_bag` extrinsic is not benchmarked	Missing benchmark, `TODO` annotation	Info
INFO_8	`PricePerByte` is set to an arbitrary low value	Missing storage deposit, `TODO` annotation	Info
INFO_9	The `rationale` parameter is not used	Unused parameter	Info

ID	Description	Category	Severity
INFO_10	The `reaction_id` parameter is not checked	Missing precondition	Info
INFO_11	The `reaction_id` parameter is not checked	Missing precondition	Info
INFO_12	Use of the `end` parameter which is unchecked	Missing precondition	Info
INFO_13	Use of the `end` parameter which is unchecked	Missing precondition	Info
INFO_14	Make the API match those of `accept_channel_transfer`	Missing precondition	Info
INFO_15	Unsafe increment of the `bounty_count`	Unsafe arithmetic	Info
INFO_16	User could be deleted by using this extrinsic	Missing precondition	Info
INFO_17	Unnecessary work in the `get_terminate_bounty_actor function`	Optimization	Info
INFO_18	Unsafe increment in `increment_active_work_entry_counter`	Unsafe arithmetic	Info
INFO_19	The worker can be the oracle	Design decision, missing precondition	Info
INFO_20	The new oracle can be a worker	Design decision, missing precondition	Info

Note

While writing this report, it was noticed that the Joystream's team also reviewed some part of the code in the scope. They noticed an issue [a] which highlights Joystream's proactive attitude towards security as well as the importance of constant code reviews to detect problems at their earliest stage.

[a] https://github.com/Joystream/joystream/issues/3698

3 Context and Scope

3.1 Context

Joystream is a Decentralized Autonomous Organizations (DAO) video platform which is controlled, owned, and operated by its users. Joystream is implemented as a standalone Substrate-based blockchain. The platform also offers its users the possibility to issue, sell and buy digital assets.

Jsgenesis, the company building Joystream, asked Quarkslab to conduct an audit on three Substrate pallets that are part of their system, namely `storage`, `content` and `bounty`.

The main concerns of Jsgenesis regarding this audit are errors that might lead to loss of funds, chain state corruptions, privilege escalation or denial of service risks.

The considered security model takes into account a misbehaving user of the Joystream ecosystem which attempts to attack the available interface surface of the parachain. The entry point for users is the JSON-RPC API which mainly exposes functions, called extrinsics.

3.2 Scope

The scope of this audit is defined by Rust modules or crates, that compose the features of a substrate-based blockchain. They are called pallets in the Polkadot ecosystem. All the audited source code is available in the main Github repository of Joystream named joystream[1].

The audit focused on the following pallets and their respective runtime configuration:

- `storage` pallet: handles the storage and distribution of content available on the platform.
- `content` pallet: handles the creation of a video channel, upload of content to the channel and the whole non-fungible token system.
- `bounty` pallet: allows the community to launch crowdfunding initiative, to incentivize people to perform a set of tasks for the common good.

More information about the specific version and the setup of the audit can be found in Section 3.3.

3.3 Audit Settings

As the Joystream codebase is undergoing significant changes with the approach of the launch of a new runtime, versions used for the audit have been frozen in agreement with the Joystream team. Exact versions and commit ID are shown in Table 3.1 for the `storage` and `content` pallets and in Table 3.2 for the `bounty` pallet. From the specified commit hashes, the runtime binary was compiled on Linux x86-64.

[1] `https://github.com/Joystream/joystream`

Project	joystream
Repository	`https://github.com/Joystream/joystream`
Commit hash	`de394c5cce0d5f0e22e4e81101604ed1e55bd2d8`
Commit date	2022/04/06
Runtime	v6.3.0

Table 3.1: joystream version references, `storage` and `content` pallets.

Project	joystream
Repository	`https://github.com/Joystream/joystream`
Commit hash	`0e188851e46f237c7216d4b676d87df3a973993a`
Commit date	2022/04/19
Runtime	v6.3.0

Table 3.2: joystream version references, `bounty` pallet.

The `joystream-node` binary compiled from this repository provides the command `joystream-node -dev -log runtime`, which is documented in the repository, to start a local test network.

4 Methodology

4.1 Familiarize with the Joystream ecosystem

The audit process started by familiarizing with the Joystream ecosystem. The Joystream Handbook[1] was the main source of documentation. The auditors took a look at the main Joystream UI for video playing[2] as well as the DAO UI[3]. They also explored the chain status using the Polkadot App UI[4]. Once they were familiar with the main Joystream concepts, they decided to mount a local dev environment, which was straight forward thanks to the scripts and instructions available in the Joystream repository.

4.2 Static code review and analysis

The next step consisted in manually reviewing the code of the pallets in order to find potential issues. Most of the attention was given to the extrinsics, which are the dispatchable functions from the blockchain. A close look was given to the internal and private functions that could be called from extrinsics and thus somehow reachable by a user.

Here is the methodology, designed as a checklist, that was used when auditing extrinsics.

Verify that weights are computed and benchmarked correctly. Weight is the computational cost of calling a dispatchable. The "pricing" of this value has to be correct to avoid cost-less execution on the blockchain which can lead to a denial of service attack. The substrate documentation recommends to write benchmarks and to run them against a specific machine configuration to measure the weight of calling extrinsics. If the weight is a constant, it must be wisely chosen via a benchmark, but a weight can also be variable according to the length or value of arguments of a dispatchable. Indeed, some extrinsics can take arrays, or blob of data, as inputs so the weight has to be adjusted to take its length into account if it can increase the computing cost.

Look for unsafe arithmetic functions. Rust is well-known for providing many protections against memory-related problems. But arithmetic, for performances reasons, is not checked when compiling in release mode. In blockchain environments, arithmetic is often critical because it is applied to assets. More generally, an overflow or an underflow could break the logic of a function or the computation of its weight, thus providing free execution or generation/destruction of assets.

[1]https://joystream.gitbook.io/testnet-workspace/
[2]https://play.joystream.org/
[3]https://dao.joystream.org/
[4]https://polkadot.js.org/apps

Look for missing storage deposits. Some extrinsics that require storage can use deposits to avoid providing free storage without asking the user to pay for important fees. Such extrinsics should be investigated because free storage means denial of service attacks.

Look for runtime panic. Parachain runtimes must be written in a defensive manner and never panic because the blockchain has to produce blocks. Thus panics must be avoided as much as possible, that is why it is interesting to look for `.unwrap()` and `.expect()`, among others, in the code.

Verify the authorization model. Extrinsics usually start with an authorization check, under the form of `ensure_something (origin)?;` or a configuration filter for example. For each extrinsic, the authorization model must be enforced in order to prevent an unprivileged user to perform a privileged operation.

Verify the usage of the transactional macro. The `#[transactional]` macro was recently introduced in Substrate and is useful in order to make sure that the side effects of an extrinsic can be reverted if it does not succeed. Otherwise, an error during a dispatchable execution could create an undetermined state for the blockchain storages.

Look for general logic and implementation errors. Look for anything that could produce an error or something that was forgotten at the time of implementation.

4.3 Configuration review

The configuration was also reviewed for the pallets that were audited. The types used by the pallets and called by the extrinsics were investigated. Most of the time, it is difficult to audit a pallet without looking at its typical configuration.

4.4 Dynamic testing

Given the weak points discovered during the static analysis phase, some test scenarios were performed to ensure that extrinsics behave correctly or that an exploitation was not possible. For that, the auditors adapted the tests that were already present in each pallet.

4.5 Tools

In addition to manual reviews and dynamic tests, some tools were used to ease the analysis. Rust Analyzer [1] was used to navigate in the code, finding correct types, and also to expand macros. Cargo-audit [2] was used to detect packages with known vulnerabilities. Also, the auditors used Clippy [3], the reference linter tool for the Rust language. For example, among others, this kind of Clippy command was used, retrieving a lot of false positive that have to be manually reviewed.

```
$ cargo clippy --no-deps -p CRATE_NAME -- -A clippy::all -W
↪  clippy::integer_arithmetic -W clippy::string_slice -W clippy::expect_used -W
↪  clippy::fallible_impl_from -W clippy::get_unwrap -W
↪  clippy::index_refutable_slice -W clippy::indexing_slicing -W
↪  clippy::match_on_vec_items -W clippy::match_wild_err_arm -W
↪  clippy::missing_panics_doc -W clippy::panic -W clippy::panic_in_result_fn -W
↪  clippy::unreachable -W clippy::unwrap_in_result -W clippy::unwrap_used
```

5 Recommendations

The following sections are organized by pallet and present the most relevant remarks for each one.

5.1 General recommendations and notes

5.1.1 Outdated dependencies

`MEDIUM_1`	Update the Substrate version		
Category	Outdated dependencies		
Status	Present		
Rating	Severity: Medium	Impact: N/A	Exploitability: N/A

The project's dependencies were analyzed using Cargo-audit [2]. This tool goes through `Cargo.lock` files and checks whether the listed packages present known vulnerabilities.

After running this tool on Joystream's codebase it was found that multiple packages, on which this project relies on, present known vulnerabilities. Many of them are related to Substrate, whose version is `v2.0.1`. Although, none of them seem to directly affect Joystream, it is recommended to upgrade to the latest version to benefit from bug fixes and, most likely, get rid of many of the warnings shown by Cargo-audit.

Warning

It is recommended to upgrade the Substrate package to its latest version, `v3.0`.

5.1.2 Use of `#[transactional]`

`INFO_1`	Use the transactional macro		
Category	Error handling		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

Substrate provides a way to treat an extrinsic in a similar way as database transactions. This is achieved by annotating extrinsics with the procedural macro `transactional`. In this way, all changes to storage performed by the annotated function are discarded if it returns `Err`, and committed if it returns `Ok`.

It is worth noting that in the code there is a visual cue, a simple line comment, which tries to deal with this situation. All checks that may result in an error are done before the cue and all stores are done after it. This is mostly used in a consistent way through all the three reviewed pallets. Although, this is a useful way to encourage to develop each extrinsic keeping in mind possible points of failure, the use of the attribute is preferred in order to avoid any mistake that can be introduced without noticing.

5.1.3 Benchmarking

Throughout the code review it was noticed that some extrinsics had a fixed weight, along with a `TODO` comment stating that the weight has to be adjusted. And, in the case of the `content` pallet, this happens for each extrinsic.

This is pointed out in the corresponding sections, farther in the report. All these issues have been rated as `Informative` because the development team is aware of the situation, as the many `TODO` items show. However, it is worth emphasizing the need to properly benchmark each extrinsic as it can lead to denial of service attacks.

Warning

Proper benchmarking is an important aspect regarding the security of the whole system. It is encouraged to tackle the mentioned issues promptly.

5.2 Storage Pallet

The `storage` pallet is responsible for managing the storage system. It provides extrinsics for manipulating storage and distribution buckets as well as functions to upload, update and delete data objects (which are mostly used by the `content` pallet).

The extrinsics provided are grouped according to the actor who carries out the action. There are two working groups that operate in the `storage` pallet:

- the storage working group;
- the distribution working groups.

Each one has its own set of workers and two separate leads, called the storage lead and distribution lead respectively.

The workers in the storage working group are called storage providers, and operate dedicated nodes for this purpose, called storage nodes. Likewise for the distribution group there is distribution providers and distribution nodes. Therefore, the extrinsics provided by this module are grouped into: storage working group leader extrinsics, storage provider extrinsics, distribution working group leader extrinsics, distribution provider extrinsics and miscellaneous extrinsics.

This is a critical pallet as it provides the means to store the content used by the whole platform. Therefore, any issue here could potentially affect every user since it might cause a denial of service on the platform.

The review of this module focused first on the extrinsics and later on the exported functions. Below are some remarks on specific functions.

5.2.1 `update_data_object_deletion_prize` extrinsic

MEDIUM_2	The `deletion_prize` parameter is not bounded		
Category	Missing precondition, partial Denial of service		
Status	Present		
Rating	Severity: Medium	Impact: Medium	Exploitability: N/A

This extrinsic updates the current `deletion_prize` for data objects. There is no check whether the new value is a valid one (neither a lower nor an upper bound check is made). It might become a potential issue in case this value is arbitrarily set. A `StorageLeader` fixing a wrong value could prevent some actions to be performed because their price would be too high for anyone to perform them or generate inconsistencies with the previous value.

INFO_2	The `update_data_object_deletion_prize` extrinsic is not benchmarked		
Category	Missing benchmark, `TODO` annotation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The weight is fixed arbitrarily to `10.000.000` (and there is no benchmark associated with this extrinsic either). However, the pallet's developers are aware of this issue since there is a `TODO` item in the code about it.

5.2.2 `update_dynamic_bag_deletion_prize` extrinsic

MEDIUM_3	The `deletion_prize` parameter is not bounded		
Category	Missing precondition, partial Denial of service		
Status	Present		
Rating	Severity: Medium	Impact: Medium	Exploitability: N/A

This extrinsic updates the current deletion prize for dynamic bags. And, similar to what happens in the `update_data_object_deletion_prize` extrinsic, there is no check on the new value.

INFO_3	The `update_dynamic_bag_deletion_prize` extrinsic is not benchmarked		
Category	Missing benchmark, `TODO` annotation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

As in the previous extrinsic, the weight is fixed (set to `10.000.000`) and there is also no benchmark associated with it. However, the pallet's developers are aware of this issue since there is a `TODO` item in the code about it.

5.2.3 `update_data_size_fee` extrinsic

MEDIUM_4	The `new_data_size_fee` parameter is not bounded		
Category	Missing precondition, partial Denial of service		
Status	Present		
Rating	Severity: Medium	Impact: Medium	Exploitability: N/A

This extrinsic updates the data size fee. There is no check on the new value set. A sudden change to a too high or too low value might bring potential issues depending on how this value is used in the system. It is recommended to set minimum and maximum values in order to avoid unexpected situations.

5.2.4 `update_families_in_dynamic_bag_creation_policy` extrinsic

LOW_1	Usage of unsafe addition		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `validate_update_families_in_dynamic_bag_creation_policy_params` function is using the `sum` function, which resorts to the regular addition defined for the type and can overflow.

LOW_2	Fixed parameter in the benchmark		
Category	Benchmark		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The benchmark function for this extrinsic uses a constant for indicating the number of distribution buckets families, called `DISTRIBUTION_BUCKET_FAMILIES_NUMBER`, which is defined in the same file. There is no extra information whether in the release version this will be the final value for the number of bucket families. Any change to this value in the release runtime would imply a wrong weight calculation.

5.2.5 `accept_storage_bucket_invitation` extrinsic

LOW_3	No check on `transactor_account_id`		
Category	Missing precondition		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

This extrinsic does not check that the `transactor_account_id` parameter is valid and matches the origin. This parameter is later used in `accept_pending_data_objects` to check the identity of the sender.

5.2.6 `create_distribution_bucket_family` extrinsic

INFO_4	Unsafe increment in `increment_distribution_family_number`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `create_distribution_bucket_family` extrinsic is calling the `increment_distribution_family_number` function which increments the count of distribution bucket families using a regular addition. However this should not pose any threat as this identifier is only incremented in this extrinsic and is of type `u64`.

5.2.7 `create_distribution_bucket` extrinsic

INFO_5	Unsafe increment in `increment_next_distribution_bucket_index_counter`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `create_distribution_bucket` extrinsic is calling the `increment_next_distribution_bucket_index_counter` function which increments the count of distribution bucket using a regular addition. However this should not pose any threat as this identifier is only incremented in this extrinsic and is of type `u64`.

5.2.8 `update_distribution_buckets_for_bag` extrinsic

LOW_4	`validate_update_distribution_buckets_for_bag_params` is missing some corner cases		
Category	Logic bug		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The computation of the `new_bucket_number` value can be wrong when $bag.distributed\_by.len().saturating\_add(add\_buckets.len()) > u64 :: MAX$. Luckily, the `add_buckets` argument is a `BTreeSet` so there cannot be any duplicates which makes the exploitation nearly impossible as the bags are then checked for existence.

LOW_5	`distribution_bucket_id` can be optimized		
Category	Optimization		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `distribution_bucket_id` are created twice, once during the check for validity of the parameters and a second time when mutating the storage. This could easily be optimized, especially if using the `#[transactional]` macro.

5.2.9 `check_buckets_for_overflow` internal function

`MEDIUM_5`	Usage of unsafe addition		
Category	Unsafe arithmetic, partial Denial of service		
Status	Present		
Rating	Severity: Medium	Impact: High	Exploitability: N/A

This function performs a raw addition for checking both the total object number and the total object size limits on a `VoucherUpdate` structure.

It is used by validation functions which are called directly or indirectly by extrinsics. It could allow to easily bypass limit checks of the storage providers, which might render some of them unresponsive by filling their storage space.

5.2.10 `compute_net_prize` internal function

`LOW_6`	Unsafe multiplication in `compute_net_prize`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `amnt` parameter is used in unsafe multiplications and can be controlled by the storage leader. This might allow to bypass a check on the available balance of a user but the exact impact is uncertain as the user would then probably drop below the existential deposit.

The `compute_net_prize` internal function is called from the `ensure_sufficient_balance` which can be reached through the content pallet.

5.2.11 Data object functions

The following functions are used by the `content` pallet to upload, update and delete content assets:

- `delete_data_objects`,
- `upload_and_delete_data_objects` and
- `upload_data_objects`.

These perform two main things: check that the provided parameters are valid and update the chain. They are generally called after the visual cue that all extrinsic have, `== MUTATION SAFE ==`, which indicates it is safe to modify the chain after that point. These methods should be used with care since calling them from the wrong location could potentially have a negative effect. In case any of these functions is called after the chain has been mutated and it fails, it could leave the chain in an inconsistent state.

A useful recommendation would be to rename them to reflect the fact that they can potentially fail. For instance, renaming `delete_data_objects` to `try_delete_data_objects`.

All these functions have something in common, they all call `try_mutating_storage_state`. This is a complex function that adds, updates and removes data objects (parametrized by the `BagOperationParams` struct) and mutates the chain to reflect the changes. No issue was found during the review of this function, however, given the extension and complexity of it, there might be corner cases which went unnoticed and that could lead to issues.

Warning

The `try_mutating_storage_state` function is a very complex function and should be kept under close attention. Although no issues have been found so far, there could be unnoticed corner cases that could lead to unexpected behavior.

5.2.12 Development mode extrinsics

There are some extrinsics grouped under the category of *development mode*, namely `sudo_upload_data_objects`, `sudo_create_dynamic_bag`, `storage_operator_remark`, and `distribution_operator_remark`. Given their name, it can be expected that these extrinsics will not be part of the release version of the runtime. However, it is worth pointing out they are currently in the codebase, and comment on some known issues regarding their weight value.

5.2.13 `sudo_upload_data_objects` extrinsic

INFO_6	The `sudo_upload_data_objects` extrinsic is not benchmarked		
Category	Missing benchmark, `TODO` annotation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The weight is fixed (set arbitrarily to `10.000.000`) and there is no benchmark associated with this extrinsic.

5.2.14 `sudo_create_dynamic_bag` extrinsic

`INFO_7`	The `sudo_create_dynamic_bag` extrinsic is not benchmarked		
Category	Missing benchmark, `TODO` annotation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The weight is fixed (set arbitrarily to `10.000.000`) and there is no benchmark associated with this extrinsic.

5.3 Content

The `content` pallet is in charge of everything related to the video platform itself. It includes:

- the creation and management of a video channel,
- posting comments on videos and their moderation,
- minting and selling non-fungible tokens of the videos.

This pallet is particularly sensitive as it handles the main features of Joystream and transfers of funds between members.

`LOW_7`	All the extrinsics have a default weight value		
Category	Missing benchmark, `TODO` annotation		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

None of the extrinsics of this pallet are benchmarked and a default weight of `10.000.000` has been set. The Joystream is encouraged to address this issue before actually using this pallet in a production runtime.

5.3.1 `create_post` extrinsic

`INFO_8`	`PricePerByte` is set to an arbitrary low value		
Category	Missing storage deposit, `TODO` annotation		
Status	Present		
Rating	Severity: Info	Impact: High	Exploitability: High

This problem is known to the Joystream team as a comment requests that this should be updated in the future.

The `PricePerByte` value is used to compute how much it costs a user to add new data into the chain. This value should reflect the actual cost for nodes and not be too low, so users can't do a denial of service on the chain by freely adding a huge amount of data to the chain.

5.3.2 `delete_post` extrinsic

`LOW_8`	The extrinsic can fail because of a race condition		
Category	Race condition		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

This extrinsic can fail due to a race condition introduced by the parameter named `witness verification`. In case another extrinsic modifying the list of posts on a video is called in the same block, the witness verification might fail.

`INFO_9`	The `rationale` parameter is not used		
Category	Unused parameter		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The rationale parameter allows moderators of a channel to state the reason why they removed a post. This parameter is not used at all by the `delete_post` extrinsic, it is not even sent in the event notifying the call to this extrinsic.

5.3.3 `react_to_post` extrinsic

`INFO_10`	The `reaction_id` parameter is not checked		
Category	Missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `reaction_id` parameter is not checked. However this parameter is not used.

5.3.4 `react_to_video` extrinsic

`INFO_11`	The `reaction_id` parameter is not checked		
Category	Missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `reaction_id` parameter is not checked. However this parameter is not used.

5.3.5 `issue_nft` extrinsic

`MEDIUM_6`	The `issue_nft` extrinsic fails on open auctions		
Category	Functionality not working		
Status	Present		
Rating	Severity: Medium	Impact: High	Exploitability: N/A

In case the non-fungible token is issued for an open auction, two opposite checks are performed so it is impossible to start an open auction. The first check is performed by the extrinsic itself to check that the non-fungible token does not exist already. The second check is performed in the `ensure_valid_init_transactional_status` function, which is called by the `construct_owned_nft` function, to ensure that the non-fungible token already exists in the case of an open auction (using the function `ensure_nft_exists`).

5.3.6 `offer_nft` extrinsic

`LOW_9`	The `to` parameter is not checked		
Category	Missing precondition		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

There is no check that the user being offered the non-fungible token exists already.

5.3.7 `buy_nft` extrinsic

HIGH_1	A logic bug could allow to steal funds from the Treasury		
Category	Logic bug		
Status	Present		
Rating	Severity: high	Impact: High	Exploitability: N/A

Note

This issue is not exploitable with the current configuration of the chain and no extrinsic allows to change that configuration.

Note

This issue is tracked on github by issue #3688[a].

[a] https://github.com/Joystream/joystream/issues/3688

A logic bug is present in the `complete_payment` function, which could potentially lead to obtaining free tokens from the Treasury. This function is used by other functions such as `buy_now` and `complete_nft_offer`, used by the `buy_nft` and `accept_incoming_offer` extrinsics, respectively.

This issue should only arise when the chain parameters `platform_fee_percentage` and `max_creator_royalty` meet the condition `platform_fee_percentage` + `max_creator_royalty` > 1. Then a malicious user could exploit it by setting up 2 accounts and selling an NFT back and forth.

The issue relates to how the royalty is paid. As it can be seen in the code below, the `match` statement has a branch where the guard clause checks whether the amount to pay exceeds the royalty plus the auction fee. In case it does, everything goes as expected (that is, the amount minus the royalty and fee is deposited to the receiver account and later, in the next statement, the royalty is deposited to the creator's account). In case it doesn't exceed the previously mentioned sum, only the amount minus the fee is deposited. However, the royalty is still paid to the creator account despite the fact that there are no funds for it. Therefore, the creator is receiving tokens at expenses of the Treasury.

```rust
/// Complete payment, either auction related or buy now/offer
pub(crate) fn complete_payment(
    in_channel: T::ChannelId,
    creator_royalty: Option<Royalty>,
    amount: BalanceOf<T>,
    sender_account_id: T::AccountId,
    receiver_account_id: Option<T::AccountId>,
    // for auction related payments
    is_auction: bool,
) {
  let auction_fee = Self::platform_fee_percentage() * amount;   // Slash amount from
↪   sender
  if is_auction {
    let _ = Balances::<T>::slash_reserved(&sender_account_id, amount);
  } else {
    let _ = Balances::<T>::slash(&sender_account_id, amount);
  }  if let Some(creator_royalty) = creator_royalty {
    let royalty = creator_royalty * amount;     // Deposit amount, excluding royalty
↪   and platform fee into receiver account
    match receiver_account_id {
      Some(receiver_account_id) if amount > royalty + auction_fee => {
        let _ = Balances::<T>::deposit_creating(
          &receiver_account_id,
          amount - royalty - auction_fee,
        );
      }
      Some(receiver_account_id) => {
        let _ =
          Balances::<T>::deposit_creating(&receiver_account_id, amount - auction_f ⌋
↪   ee);
      }
      _ => (),
    };    // deposit to creator account
    ContentTreasury::<T>::deposit_to_channel_account(in_channel, royalty);
  } else if let Some(receiver_account_id) = receiver_account_id {
    // Deposit amount, excluding auction fee into receiver account
    let _ = Balances::<T>::deposit_creating(&receiver_account_id, amount - auction_ ⌋
↪   fee);
  }
}
```

This bug can be triggered by updating the `platform_fee_percentage` and `max_creator_royalty` chain parameters. Running the test `buy_nft` and printing the balance of the involved users allows to see this bug in action.

In order to fix this bug, the first step is to perform some kind of check (either enforced in the code or documented clearly) which prevents the sum of those parameters to exceed the maximum possible value (that is, 100%). Then a design decision is necessary to determine what to do in case there is not enough funds to pay both for the royalties and the fees. For example, deciding if paying the owner of the NFT or paying the royalties should be prioritized. In the latter case, the owner should always receive `amount.saturating_sub(royalties).saturating_sub(auction_fee)` which will prevent the issue.

MEDIUM_7	The payment for a non-fungible token can go to the wrong user		
Category	Wrong postcondition, fund loss		
Status	Present		
Rating	Severity: Medium	Impact: High	Exploitability: N/A

Note

This issue is fixed in pull request #3654[a].

[a] https://github.com/Joystream/joystream/pull/3654

The account id receiving the payment is determined using the following line:

```
let owner_account_id = ContentTreasury::<T>::account_for_channel(video.in_channel);
```

However, the owner of the channel is not necessarily the owner of the NFT. Therefore, the tokens can be deposited in the wrong account. This is handled correctly in the function `complete_auction` which is using the `ensure_owner_account_id` function to determine who should receive that payment.

LOW_10	Use of `can_slash`		
Category	Postcondition can be false		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `ensure_sufficient_free_balance` function should not use the `can_slash` as, per the documentation "NOTE: slash() prefers free balance, but assumes that reserve balance can be drawn from in extreme circumstances".

5.3.8 `start_english_auction` extrinsic

LOW_11	The `end` parameter is not checked		
Category	Missing precondition		
Status	Present		
Rating	Severity: Low	Impact: Low	Exploitability: N/A

The `end` parameter is used by other extrinsics such as `make_english_auction_bid` to make sure the auction is still in progress but this parameter is not checked. This parameter could be removed and computed from the `duration` parameter as the information is redundant.

5.3.9 `make_english_auction_bid` extrinsic

`MEDIUM_8`	Funds can stay locked forever		
Category	Missing postcondition, fund loss		
Status	Present		
Rating	Severity: Medium	Impact: High	Exploitability: N/A

Note

This issue is fixed in pull request #3592 [a].

[a] `https://github.com/Joystream/joystream/pull/3592`

In case someone other than the top bidder makes a higher bid than the previous top bid, the previous top bidder's bid is not unreserved.

`LOW_12`	Top bidder requires more funds to bid again		
Category	Logic bug		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

In case the top bidder wants to bid over his previous bid, he needs to have the full amount in his `free balance` while the difference between the old bid and the new bid amounts should be sufficient.

`LOW_13`	Check in `ensure_constraints_on_bid_amount` should be >=		
Category	Logic bug		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `ensure_constraints_on_bid_amount` is checking the condition $amount > buy\_now$ which is not consistent with the rest of the code in the pallet. Moreover, if the top bid is too close to `buy_now_price`, the extrinsic will fail if a user sends a bid of `buy_now_price`.

INFO_12	Use of the `end` parameter which is unchecked		
Category	Missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `make_english_auction_bid` extrinsic is calling the `ensure_auction_is_not_expired` function which is using the `end` parameter which is not checked during the auction creation.

5.3.10 `claim_won_english_auction` extrinsic

INFO_13	Use of the `end` parameter which is unchecked		
Category	Missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `claim_won_english_auction` extrinsic is calling the `ensure_auction_is_not_expired` function which is using the `end` parameter which is not checked during the auction creation.

5.3.11 `accept_incoming_offer` extrinsic

INFO_14	Make the API match those of `accept_channel_transfer`		
Category	Missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The API of `accept_incoming_offer` extrinsic should be changed to match the API of `accept_channel_transfer`, both for coherence and prevent future updates to open the possibility of a race condition. If the `offer_nft` function did not check that the status is `Idle` or if a function allowed to update the offer, the `accept_incoming_offer` extrinsic would be vulnerable to a race condition (updating the offer in the same block would make the other user accept a different price than expected).

5.3.12 `accept_channel_transfer` extrinsic

MEDIUM_9	A logic bug that could allow previous channel collaborators manipulate its assets		
Category	Logic bug		
Status	Present		
Rating	Severity: Medium	Impact: MEDIUM	Exploitability: N/A

Note

This issue is fixed in pull request #3704[a].

[a] https://github.com/Joystream/joystream/pull/3704

The `update_channel_transfer_status` extrinsic allows an authorized actor to change the owner of a channel along with its collaborators (changing the status of the channel temporarily from `NoActiveTransfer` to `PendingTransfer`). In order for the transfer to complete, the new owner has to call the `accept_channel_transfer`. There is a bug in the latter where the new set of collaborators is never updated (that is, it leaves the old set unchanged). This could lead to potential issues with the management of the channel since collaborators are authorized to do relevant actions to the channel (e.g., updating its assets).

5.3.13 `update_max_reward_allowed` extrinsic

LOW_14	Missing boundary check		
Category	Boundary check		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

This extrinsic is used to set a new value for the maximum allowed reward, which is used by `claim_channel_reward`. The new value is stored without checking it is within valid boundaries (both lower and upper bounds).

5.3.14 `update_min_cashout_allowed` extrinsic

LOW_15	Missing boundary check		
Category	Boundary check		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

This extrinsic is used to set a new value for the minimum allowed cash out, which is used by `claim_channel_reward` The new value is stored without checking if it is within valid boundaries (both lower and upper bounds).

5.3.15 Multiple race conditions on extrinsics

LOW_16	Multiple extrinsics can fail because of race conditions		
Category	Race condition		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `cancel_offer`, `cancel_buy_now`, `update_buy_now_price`, `buy_nft`, `accept_incoming_offer` and `cancel_english_auction` extrinsics can all fail due to race conditions.

Both `cancel_buy_now`, and `update_buy_now_price` can fail if `buy_nft` is called in the same block. The opposite effect, making `buy_nft` fail is also possible, depending on which extrinsic will be processed.

The same issue arises with `cancel_offer` and `accept_incoming_offer`.

The extrinsic `cancel_english_auction` will fail if a bid is made in the same block.

5.3.16 Channel transfer status

Channels can be transferred between members. This is achieved using the `update_channel_transfer_status` and `accept_channel_transfer` extrinsics. To keep track of this a `transfer_status` flag is used which can be either `NoActiveTransfer` or `PendingTransfer`. When a transfer is initiated, `transfer_status` is set to the latter, which also holds the parameters of the transfer (new owner and set of collaborators).

`LOW_17`	Possible inconsistent check of the `NoActiveTransfer` flag across extrinsics		
Category	Inconsistent usage		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

Many extrinsics check the aforementioned flag, making sure there is no active transfer, before performing actions on a channel. Examples of this are `update_channel` and `update_moderator_set`. In this case, no changes can be made to the channel until the transfer is done. However, there are other extrinsics, such as `delete_channel` where there is no checks regarding this flag.

`LOW_18`	Possible missing parameter when transferring a channel		
Category	Missing parameter		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `update_channel_transfer_status` extrinsic is used to do a channel transfer which updates the owner and collaborators of the given channel. However, the moderator set is not included among the parameters to update. This can have negative effects since moderators can modify a channel and its assets using extrinsics such as: `delete_channel_assets_as_moderator` and `delete_channel_as_moderator`. It is worth noting that the new owner can update the moderator set after the transfer is completed.

5.4 Bounty

The `bounty` pallet is handling the bounty feature which is a crowdfunding mechanism that allows individuals or the council to reward a list of tasks `workers` will have to perform. A bug in the pallet can become critical as it would likely allow to steal funds. For example, an attack scenario that was considered is that if a user was able to make a bounty fail during the funding stage, it would allow him to steal the cherry of every bounty (the cherry is a way to reward funders if the bounty fails to reach the target funding).

5.4.1 `create_bounty` extrinsic

`LOW_19`	No upper bound for `funding_period`		
Category	Missing precondition		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

There is no maximum value being checked for the `funding_period` for a `FundingType::Limited` bounty. The `funding_period` is later used in an unsafe addition in the `BountyStageCalculator`.

`INFO_15`	Unsafe increment of the `bounty_count`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `bounty_count` is incremented using a regular addition. However this should not raise any threat as this identifier is only incremented in this extrinsic and is of type `u64`. The consequence would be to override the first bounties that were submitted.

`INFO_16`	User could be deleted by using this extrinsic		
Category	Missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

This could potentially be problematic if the user creating the bounty also nominated himself as the oracle.

5.4.2 `fund_bounty` extrinsic

`LOW_20`	Unsafe arithmetic in `funding_period_expired`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The fund_bounty extrinsic is calling the funding_period_expired function of the BountyStageCalculator which is making an unsafe addition with the funding_period parameter of the bounty which does not have an upper bound.

5.4.3 terminate_bounty extrinsic

INFO_17	Unnecessary work in the get_terminate_bounty_actor function		
Category	Optimization		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

As shown in the snippet below, where the function ensure_bounty_actor_manager called by get_terminate_bounty_actor has been manually inlined, the ensure_bounty_actor_manager is already doing all the necessary work.

```
fn get_terminate_bounty_actor(
    origin: T::Origin,
    bounty: &Bounty<T>,
) -> Result<BountyActorManager<T>, BadOrigin> {
    let bounty_creator_manager = match bounty.creation_params.creator {
        BountyActor::Member(member_id) => {
            let account_id =
                T::Membership::ensure_member_controller_account_origin(origin, member_i
↪   d)?;

            Ok(BountyActorManager::Member(account_id, member_id))
        }
        BountyActor::Council => {
            ensure_root(origin)?;

            Ok(BountyActorManager::Council)
        }
    }

    let actor = match bounty_creator_manager {
        Ok(creator_manager) => creator_manager,
        Err(_) => {
            ensure_root(origin)?;
            BountyActorManager::Council
        }
    };

    Ok(actor)
}
```

5.4.4 `announce_work_entry` extrinsic

`LOW_21`	Unsafe increment of the `entry_count`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

The `entry_count` is incremented using a regular addition. This identifier is shared by all bounties so it is more dangerous. However as it is of type `u64`, this increment should not raise any threat in the medium term.

`INFO_18`	Unsafe increment in `increment_active_work_entry_counter`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `increment_active_work_entry_counter` is incremented using a regular addition. However this should not raise any threat as this identifier is a `u64` and this extrinsic has a proper weight. It should not be possible to have such a huge number of contributions and the impact would be low.

`INFO_19`	The worker can be the oracle		
Category	Design decision, missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

This extrinsic does not check if the member announcing a work entry is the same as the oracle. If this design decision is intended, which is coherent with the fact that the oracle is supposed to be trustworthy, it should be stated explicitly.

5.4.5 `switch_oracle` extrinsic

`INFO_20`	The new oracle can be a worker		
Category	Design decision, missing precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

This extrinsic does not check if the new oracle already announced a work entry. If this design decision is intended, which is coherent with the fact that the oracle is supposed to be trustworthy, it should be stated explicitly.

5.4.6 `submit_oracle_judgment` extrinsic

`MEDIUM_10`	Funds of participants can stay locked		
Category	Missing precondition/postcondition		
Status	Present		
Rating	Severity: Medium	Impact: N/A	Exploitability: N/A

In case the judgment does not list all participants, the omitted participants funds will stay locked even after the bounty is destroyed. The extrinsic is either missing a precondition, stating that all participants need to be listed in the judgment, or a postcondition to perform a cleanup on all the remaining work entries after the judgments triage loop.

`LOW_22`	Missing event in case of reject		
Category	Missing postcondition		
Status	Present		
Rating	Severity: Low	Impact: N/A	Exploitability: N/A

In case a work entry is rejected by the oracle, no event is sent to notify of the slashing of the worker's stake.

5.5 Runtime Configuration

No major issues were found during the review of the configuration. Some parameters are set to default values that need to be changed. These values are annotated with a `TODO` comment.

6 Conclusion

Jsgenesis, the company building Joystream, asked Quarkslab to conduct an audit on three Substrate pallets that are part of their platform, namely `storage`, `content` and `bounty`.

The main concerns of Jsgenesis regarding this audit are errors that might lead to loss of funds, chain state corruptions, privilege escalation or denial of service risks.

The audit exposed some medium to high-level issues as well as many low level and informative ones. Some of these issues, which were considered important, were disclosed to the developers, who promptly worked on fixing them.

Each reviewed pallet posed its own challenge. Mostly from the business logic perspective since each provide complex functionality.

The `storage` pallet, which provides the means to store and keep track of data objects, is a key component of the platform. Any issue in it could potentially affect the entire platform from a denial-of-service perspective. The code review showed some issues, although none critical, and identified a complex and extensive function at the core of many others which should be watched closely in order to avoid unexpected behavior in the future.

The `content` pallet, besides providing the means to add, update and remove content from the platform (and which makes extensive use of the `storage` pallet), also offers the possibility to issue, sell and buy digital assets. Therefore, it is an important module since it manipulates funds and it is a potential target for any malicious actor. This pallet presented the most important issues found during the audit. It is worth mentioning that this module is still under development and there are important details to tackle, such as properly benchmark each extrinsic.

The audit ended with the `bounty` pallet, responsible for creating and managing bounties. Bounties are the platform's mechanism to get tasks done (such as uploading videos, translating content, etc.). Either members or the Council can create a bounty for a particular task and assign a reward to it. The lifetime of a bounty is complex, it goes through many stages and different actions can be carried out on each stage. Despite this complexity, the code almost presented no issues.

Quarkslab encourages Joystream to adopt the recommendations listed in this report and to continue having their code audited by external companies.

Finally, Quarkslab would like to highlight Joystream's team responsiveness when asking questions or reporting issues.

Glossary

cargo-audit A tool for auditing `Cargo.lock` files for crates with security vulnerabilities reported to the RustSec Advisory Database. See `https://crates.io/crates/cargo-audit`.

clippy A collection of lints to catch common mistakes and improve your Rust code. See `https://github.com/rust-lang/rust-clippy`.

extrinsic State changes that come from the outside world, i.e. they are not part of the system itself. Extrinsics can take two forms, "inherents" and "transactions".

non-fungible token A Non Fungible Token is a unique unit of data that can be sold or traded.

pallet Substrate modules exposing various extrinsics, events, errors and storage items that will be compiled in the runtime and usable by users or other components. It is implemented as Rust crates.

Acronyms

DAO	Decentralized Autonomous Organizations.

Bibliography

[1] Rust-analyzer. *Rust-analyzer*. URL: `https://github.com/rust-analyzer/rust-analyzer` (visited on Mar. 14, 2022) (cit. on p. 9).

[2] rustsec. *cargo-audit*. URL: `https://crates.io/crates/cargo-audit` (visited on Mar. 14, 2022) (cit. on pp. 9, 11).

[3] rust-lang. *rust-clippy*. URL: `https://github.com/rust-lang/rust-clippy` (visited on Mar. 14, 2022) (cit. on p. 9).