Securing Forgejo Actions Deployments

- Forgejo Access
- Runner Registration
  - Ephemeral Runners
- Preventing Arbitrary Workflow Execution
- Runner Configuration
  - Job Containers with Docker
    - container.privileged
    - container.network
    - container.valid_volumes
    - container.docker_host
    - Resource Constraints via container.options
    - Private Registry Access
  - Job Containers with LXC
  - Execution on Host (host)
  - runner.timeout
- Local Network Resources
- Patching

Remote code execution (RCE) is typically considered one of the most critical security vulnerabilities that can be discovered and exploited in a system. For Forgejo Actions, their entire purpose is to perform remote code execution. Therefore, there are many security considerations in deploying a self-hosted Forgejo Runner.

Every administrator of a self-hosted Forgejo Runner will have a different set of requirements for their deployment to be successful, a different risk environment in which they are deploying their Runner, and a different appetite to accept risk. They will need to make trade-offs to meet their goals and still be able to sleep soundly at night. This documentation will not outline one perfect solution, but it will attempt to inform any administrator about the known risks, and the tools that

they can use to manage those risks.

Let's assume that a threat exists, and then we'll explore how we can prevent it. Our threat is: a malicious user, Mallory, is attempting to access a Forgejo instance, and Mallory intends to run customized Forgejo Actions workflows in order to extract confidential data, plant viruses and worms in build artifacts and code, and then crash the system to make it more difficult to recover from their attack.

How can we make Mallory's life really difficult?

Forgejo Access

The first thing Mallory needs to do is access the Forgejo instance that they want to attack, and login to the system.

Many Forgejo instances are deployed publicly in order to enable the distribution of Open Source software. But for some systems, it may be advisable to prevent public access altogether. When deploying a self-hosted Forgejo instance, it can be deployed so that it can only be accessed by users on a local network or a virtual private network (VPN). If deployed like this, Mallory's job is more difficult: they need to escalate into the target network before they can get started. But this configuration isn't appropriate for everyone, and it is just one layer in protecting the instance from Mallory.

After network access is established, managing the risks to a Forgejo Runner system continues with managing the users that have access to the system and the repositories that they can work upon.

- `service.DISABLE_REGISTRATION` can be set to `true` which will prevent unexpected users from registering on the Forgejo instance. Mallory will not be able to create their own Forgejo account. Forgejo administrators can create new accounts for users on-demand.
- `repository.MAX_CREATION_LIMIT` can be set to `0` in order to prevent users from creating unexpected repositories on the instance. Mallory won't be able to create new empty repositories from which they may be able to run malicious workflow actions. Forgejo administrators can override this value temporarily when new authorized repositories are created, allowing the system to be usable despite this configuration.
  - For any value other than `-1` and `0`, be aware that `admin.DISABLE_REGULAR_ORG_CREATION` must also be set to `true` to prevent users from effectively bypassing this limitation by creating organizations.
- `repository.ALLOW_FORK_WITHOUT_MAXIMUM_LIMIT` can be set to `false`, which will prevent users from forking existing repositories. Mallory won't be able to fork existing repositories, preventing them from creating pull requests that can trigger workflow runs.

There is no ideal configuration for these values, as changing them affects Forgejo access and security, but limits functionality.

It's easy to assume that a user on a Forgejo instance is an authorized user of that instance. In close-knit instances of known users, administrators should review `Admin settings` > `Identity & access` > `User accounts`, and manually ensure that 2FA is enabled for users. This reduces the risk of Mallory using an old leaked password from a data breach in order to masquerade as an authorized user account.

Runner Registration

We've reviewed a few ways to keep Mallory out of the Forgejo system entirely, but you might choose to invite Mallory into the Forgejo instance. They could be an Open Source contributor in your wonderfully open project, or they could be a new employee in your commercial software organization. We still have ways to minimize risk from the threat that they pose.

Forgejo Runner instances can be registered with Forgejo at four different levels, which will impact the jobs that they will run. By ensuring that the tightest registration is used, the scope of risk from Mallory is reduced:

- **Repository** - A runner registered with a repository will only run actions defined in that repository.
- **Organization** & **User** - A runner registered with an organization or user will only run actions defined in a repository owned by that organization or user.
- **Global** - A runner registered in Forgejo's site administration can run a job in any repository.

Ephemeral Runners

Administrators of environments where Forgejo Runner instances are created on demand should consider using Forgejo Runner in ephemeral mode. That is especially true when running workflows in the same virtual machine or container without isolation (`host`) because the runner token is then accessible to the running job. When in ephemeral mode, as opposed to persistent mode, which is the default, Forgejo will assign a runner at most one job before removing it. That means that even if a runner token falls into Mallory's hands, Mallory will not be able to request any other job.

Ephemeral mode can be enabled during runner registration by passing `--ephemeral` to `forgejo-runner register` or `forgejo forgejo-cli actions register`. The HTTP API exposes an `ephemeral` property, too. Ephemeral mode is enforced by Forgejo and cannot be disabled or ignored.

Forgejo will only remove ephemeral runners after they have received one job and completed it or exceeded a timeout. That means that ephemeral runners can continue asking Forgejo for a job until one is available and potentially linger forever if they do not receive any job. There is currently no way to influence that behavior.

With ephemeral runners, only `forgejo-runner one-job` can be used. `forgejo-runner daemon` will exit immediately when it is told by Forgejo to switch to ephemeral mode.

Preventing Arbitrary Workflow Execution

As an administrator of Forgejo Runner, if you wish to ensure that a malicious user like Mallory can never run unauthorized workflows, some of the responsibility will fall onto administrators of the Forgejo instance and the repository administrators. That might also be you, or it might include other colleagues and collaborators. Let's assume that a Forgejo Runner is configured on a specific repository, the tightest registration described in Runner Registration, and explore what we'd need to know to avoid Mallory running an unauthorized workflow.

If Mallory can push directly to the repository, they can run arbitrary workflows. Branch protection rules may appear to be a line of defense (for example: workflows are defined in `main` and Mallory can only create branches & PRs, not modify `main`), but Mallory has workarounds for this if they can push to any new branch: they can create a new branch that isn't covered by branch protection, with a workflow with a matching `on: push` event trigger, and Mallory will be able to run any workflow that they want.

If Mallory does not have direct write access, but can fork the repository and create pull requests, then their first pull request to the repository will require review from a contributor and approval before workflows can execute. After that, any future pull request that Mallory makes will have the workflow executed, including any modifications made in the fork. This can include the creation of a new workflow that has the `on: pull_request` event, allowing them to introduce workflows that never existed before. The `on: pull_request` workflows that would be executed in this model do have additional restrictions relative to other workflows -- they have no access to the secrets in the repository.

If the repository has an `on: pull_request_target` workflow defined in it, then Mallory cannot modify the code that is executed in that workflow unless their pull requests are merged -- the workflow runs with the content of the target branch of the PR, rather than with the content of the pull request. However, these workflows do have access to secrets, and can sometimes make mistakes in interacting with the code in the pull request in a dangerous manner. `on: pull_request_target` workflows require careful engineering and review to ensure that they do not interact with other code in the repository to introduce new security risks -- these considerations are described in `on.pull_request_target`.

Runner Configuration

Once Mallory gets access to a repository where they can begin mutating actions workflows, and a runner is configured which will execute those workflows, their access to resources varies considerably based upon the configuration of that runner.

When registering a Forgejo Runner, an administrator will be prompted for runner labels which define the environment that jobs will be executed in. Each of the different label types (`docker`, `lxc`, and `host`) will have different impact on the environment which Mallory would be able to access.

Job Containers with Docker

When Forgejo Runner has labels configured to use a Docker daemon, many of the configuration options in the `container` section of the Runner's config are security sensitive. The default values are chosen to minimize security risks, but any changes to these values require the administrator to be aware of security side-effects.

Multiple different containers can be used in a single job. These terms will be used to describe each one:

- **Job containers**: the container image defined by the runner label, or overridden by the `jobs.<job_id>.container.image` property, from which future step commands are executed.
- **Service containers**: containers defined in the `jobs.<job_id>.services` section of the workflow file.
- **Step containers**: containers that are required to `jobs.<job_id>.steps.uses` which use a "Remote container action" or a "Remote action" that is defined with `uses: docker`.

`container.privileged`

An administrator can change `container.privileged` from the default value of `false` to `true`. When it is configured as `true`, each of the job containers, service containers, and step containers will be launched with the `--privileged` flag:

The `--privileged` flag allows broad access to the host system through elevated runtime privilege and Linux capabilities.

If `container.privileged` is configured to `true` and our attacker Mallory is able to mutate actions workflows that are executed, Mallory will be able to operate as `root` on the Forgejo Runner machine; all confidential data can be compromised, all data integrity can be compromised, and availability of the service can be disrupted.

Step containers may be defined by remote action workflows, which rely on the integrity of workflows defined in other repositories which may even be hosted in other git forges. If those remote workflows are compromised, running them with privileged access will cause risks to the host that bypass some of the Forgejo access restrictions; Mallory doesn't need to access your Forgejo instance at all if they can mutate a remote workflow that is being executed. Users authoring workflows can minimize the risk of this by using sha-pinned references and reviewing changes when they occur carefully. As an administrator there are no direct controls to prevent this, and it would be inadvisable to use `container.privileged` set to `true` if you believe this is a risk and don't take any further isolation action.

`container.network`

An administrator can change `container.network` from the default value of `""` to `"host"`, `"bridge"`, or a custom network name. The default value `""` causes a new isolated network to be created for each new job, and service/step containers are created on the same network to facilitate communication between them.

Changing the network mode to `"host"` will remove restrictions to allow all containers to communicate freely with each other and the host, without any job isolation. This can be a risk to the confidentiality of data hosted by other services on the host; for example, if a web server is running on `127.0.0.1:3000` that would normally be protected by access controls imposed by a reverse proxy like nginx, Mallory will be able to run job containers that can access this web server directly bypassing those controls. This can risk the availability of workflow jobs as well, by sharing the same network stack; if Mallory starts a workflow which starts a database server that listens to port `5432` which is already in-use in the host, it would probably fail to start due to a network port conflict, but they may be able to claim the port during database server maintenance and affect the availability of the other service.

Changing the network mode to `"bridge"` will allow unrestricted network access, but scoped only to other docker containers running on the same default `"bridge"` network. This is more restricted than `"host"`, but allows similar risks if there are confidential services running in docker that are outside the job container.

Changing the network mode to a custom network name will isolate the network of all job containers, but they will not be isolated from each other. In combination with a `runner.capacity` configured `> 1` where multiple workflows will be executing concurrently on the same custom named network, the availability of workflow jobs can be affected as container names can conflict with each other; for example, if two jobs run a service container named `pgsql`, DNS resolution for the name `pgsql` will round-robin between the two containers. Confusion between intermittent workflow failures may be the most likely outcome, but our attacker Mallory may be able to access unexpected data through network access to other job, service, or step containers.

`container.valid_volumes`

The default value of `valid_volumes` is an empty array `[]`. If an administrator changes this, they will allow a job container or service container to mount the listed volumes when the container is started.

Data on the mounted volumes can be accessed freely by the Forgejo Actions, in both read and read/write mode. It is trivial to execute commands as `root` within the containers as there is no access control at this level, so no filesystem-level permissions will be effective in blocking access.

If a volume is listed in `valid_volumes`, the data within it cannot be considered confidential as Mallory would be able to read the data through mutated workflow actions. It also cannot safely be used as an input to critical build jobs without having signature or integrity checks performed against an external validation, as Mallory can mutate this data and may be able to sneak malicious code into build artifacts derived from the volume contents.

`container.docker_host`

`container.docker_host` serves two purposes in the runner configuration; both defining where the runner will execute containers, and defining whether that same access to container execution will be shared with the containers. More detailed information on this configuration parameter is provided in the Utilizing Docker within Actions document, along with the security risks associated with it.

Resource Constraints via `container.options`

Typically, job, service, and step containers are executed with no resource constraints, which allows Mallory to consume as much of the host's CPU and memory as they can. This can affect the ability of the runner server to serve future tasks, to operate continuously, and if the resources are shared with other services, affect the ability to provide those shared services.

The `container.options` value can be used to define docker container resource limitations such as `--memory=1g` (limiting the container to 1 GB), or `--cpus=2` (limiting the container to 2 CPU cores). When combined with `runner.capacity`, this allows you to constrain the resources used by job containers and step containers.

However, there is no limit in the Forgejo Runner on the number of simultaneous service containers that a single workflow job can start. There are also no mechanisms available to limit disk or network I/O for any container.

Private Registry Access

If Docker or Podman is logged into OCI image registries that require authentication, then Forgejo Runner can pull images from those registries, too. Workflow authors can specify arbitrary images in workflows using the `container:` key. Therefore, they can potentially use any image Docker and Podman have access to.

If that is not desired, ensure that Docker and Podman are not logged into any registry that requires authentication.

Job Containers with LXC

When using `lxc` labels in Forgejo Runner, there are no mechanisms available to restrict resource utilization (memory, CPU, disk & network I/O), which would allow Mallory to consume high levels of resources and affect the availability of the host for other workflow executions, and other shared services.

Execution on Host (host)

When using the `host` label to allow jobs to run without containers, step commands are executed as the same user as the Forgejo Runner is executed as, and in the same host without any isolation.

Mallory's malicious workflow executions will be able to perform any work on the host that the Forgejo Runner's user would be able to perform. If the Forgejo Runner is executed under an init system like `systemd` and the systemd unit is hardened and uses a dynamic user, various restrictions can be imposed upon the daemon by the init system. Unfortunately these limitations are outside of the scope of this document as they are specific to the init system being used. However, specific risks to note are:

- The host's network stack will be used by any executed commands. This can risk the confidentiality of other services on the host; for example, if a web server is running on `localhost:3000` that would normally be protected by access controls imposed by a reverse proxy like nginx, Mallory will be able to run job containers that can access this web server directly bypassing those controls. This can risk the availability of workflow jobs as well by sharing the same network stack; if Mallory starts a workflow which starts a database server that listens to port `5432` which is already in-use in the host, it would probably fail to start due to a network port conflict, but they may be able to claim the port during database server maintenance and affect the availability of the other service.
- There are no mechanisms within Forgejo Runner to restrict resource utilization (memory, CPU, disk & network I/O), which would allow Mallory to consume high levels of resources and affect the availability of the host for other workflow executions, and other shared services. Hardening with the init system should be able to provide some restrictions if needed.

- The Forgejo Runner necessarily needs to read the runner's `.runner` state file. In the host execution mode, a workflow job would also be able to read this state file. If Mallory exfiltrated this state file, they would be able to set up an impersonating Forgejo Runner. As Forgejo fed jobs to this runner's polling routine, the confidentiality of secrets would be exposed to the impersonating Forgejo Runner.

`runner.timeout`

Creating long-running workflow jobs would allow Mallory to impact the availability of Forgejo Runner for other users. For example, the default 3 hour job timeout and `runner.capacity` of `1` would allow a simple bash loop to prevent other jobs from running for 3 hours. These jobs can be cancelled if observed.

Local Network Resources

Regardless of all the configuration options we've reviewed so far, as long as Mallory is able to execute arbitrary workflows that they author, you're exposing local network resources to interaction from Mallory's changes. For example, if your Forgejo Runner instance is colocated on the same network as end-user laptops and desktops, Mallory could enumerate local network resources through subnet scanning and attempt to attack those machines over the network.

If desired, preventing this type of attack will require isolating the subnet that the Forgejo Runner machine is hosted upon, and then limiting cross-subnet access at a firewall. If the Forgejo Runner is hosted on a virtual machine, tagged VLANs can be enforced at the VM level in order to perform this configuration without requiring physical network changes. The specifics of these configurations are out-of-scope for this document as they depend on the wide variety of network configurations and equipment in the wild.

Patching

All security configurations degrade over time as vulnerabilities are discovered and exploited in the software that are used to host services. If a security vulnerability is discovered in any component of the system that Forgejo Runner is hosted upon, the expected protection won't be present. Ensure that your operating system and other software on the host is kept up-to-date with upstream security releases.

Forgejo and Forgejo Runner have security releases from time-to-time as new issues are discovered. In order to be informed when these releases are scheduled, you can subscribe to notifications from the forgejo/security-announcements repository on Codeberg.

□□□ #2
□ □□□ □□□ 2026-10-31 13:33:14 UTC
□ □□□ □□□ 2026-10-31 13:33:16 UTC