RCE Administration and Security Guide

Build 10.3.1.202202230837

Table of Contents

List of Figures

Chapter 1. Preface

This document is the main Administration and Security Guide for RCE. It is meant to provide practical information for IT administrators, and also provide background information regarding the security aspects of RCE setups for inspection and consideration. It is currently limited in scope and will be expanded over time.

Chapter 2. RCE Administration

2.1. Installation as a Service on a Linux Server

For ad-hoc or temporary RCE network setups, running a headless RCE from the command line is perfectly fine. For more permanent installations, however, we recommend installing RCE as a Linux systemd Service instead. This has the advantage that RCE automatically shuts down when the server is shut down, and automatically restarts when the server does.

2.1.1. RCE as a Linux systemd Service

This section describes setting up RCE as an auto-start service on Linux.

Please note that the "`rce-daemon`" command provided with previous RCE releases for managing System V services is *deprecated*, and will most likely be removed in RCE 11.0.

Execute these steps to configure an RCE instance to run as a systemd service:

- If not already present, install RCE using the .deb or .rpm package. (While installing a service from the plain .zip distribution is possible, the extra steps regarding file locations and permissions are not covered by this guide.)
- Choose a *command-line id* for the instance; in the templates below, this is marked as `<ID>`. Ideally, it should only consist of ASCII letters and numbers to prevent command-line complications.
- Choose a *human-readable description* for the instance; in the templates below, this is marked as `<your_description>`. Unlike the ID, this is much more flexible regarding allowed characters. This description is shown when listing services using systemd commands.
- Choose a *user and group* to run the RCE instance under. Both the user and the group must already exist. In the template below, replace `<User>` and `<Group>` with the respective names (e.g. "user1") or numeric ids (e.g. "1000").
- Choose a *profile folder*; in the template below, this is marked as `<profile_folder_name>`. In a standard file system layout, the instance's full profile path will be `/home/<UID>/.rce/<profile_folder_name>`. This folder does not need to exist, as RCE will create it automatically when that profile is first used (as long as the chosen user/group has sufficient permissions to do so).
- Copy the template below into a file named `rce-<ID>.service` in `/etc/systemd/system/`. Note that on typical systems, you will need root/sudo permissions to create or edit this file.
- By default, make sure this file's permissions are `0644` and ownership is `root:root`.
- Replace all <placeholders> (description, UID/GID, and the profile folder) in that file.
- **Important:** execute the command `systemctl daemon-reload` (as root or via sudo) after editing the service file to apply the changes.
- In systemd terminology, the created .service file is called (or represents) a "service unit", which is then controlled through various standard commands. See the sect2 "Useful management commands"

below for a list of commands. For now, just use `systemctl start rce-<ID>` to start the instance, and `systemctl status rce-<ID>` to see if it started as expected. If it did, use `systemctl enable rce-<ID>` to auto-start the service whenever the host machine is rebooted.

The .service unit file template

Lines ending with a backslash ("\") are only wrapped for layouting purposes; these can also be merged into single long lines.

```
[Unit]
 Description=<your_description (without quotes)>

 After=network-online.target
 Wants=network-online.target

 [Service]
 User=<User>
 Group=<Group>
 ExecStart=/usr/bin/rce -p <profile_folder_name> \
   --headless --launcher.suppressErrors -nosplash
 ExecStop=/usr/bin/rce -p <profile_folder_name> \
   --shutdown --launcher.suppressErrors -nosplash
 Type=simple

 [Install]
 WantedBy=multi-user.target
```

Useful management commands

These commands must be either run as "root" or prefixed with "sudo".

- systemctl start rce-<ID> - starts the service
- systemctl stop rce-<ID> - stops the service
- systemctl restart rce-<ID> - unsurprisingly, restarts the service
- systemctl status rce-<ID> - display the service's status and its latest output
- systemctl enable rce-<ID> - marks the service to auto-start after a reboot
- systemctl disable rce-<ID> - disables auto-start after a reboot
- systemctl list-units "rce-*" - lists all installed RCE services

2.1.2. Daemon Configuration

After installation, the daemon instance will be started automatically. This will create a default configuration file if it does not exist yet.

To configure the daemon instance edit and save the configuration file (*configuration.json*) that is located in the *profile folder*. Then use `systemctl restart rce-<ID>` to apply the new configuration.

For importing SSH credentials and authorization group keys into a daemon, please refer to section "Importing authorization data without GUI access" in the User Guide. As of RCE 10, file-based imports are only processed on startup, so a restart is required for this, too.

Note

The need to restart the daemon is temporary; future versions of RCE will apply configuration changes as soon as configuration files are changed or new import files are placed in the respective folders.

2.2. Setting up Cross-Organization Networks using RCE Uplink

"RCE Uplink" is an experimental feature introduced in RCE 10.x that addresses the growing need for the exchange of computation services between different organizations. Conceptually, this is realized using RCE's standard approach of providing access to local tools as distributed services, while keeping the tool's files and execution on the local machine.

To allow the efficient and low-overhead exchange of tool computation services between organizations, some sort of network connection must be established between them. For IT security reasons, however, many organizations are understandably reluctant to open ports into their networks. The Uplink approach addresses this by providing a shared coordination and forwarding server called a "relay". This relay server is typically placed outside of any organization's protected network, e.g. on a rented server or in the DMZ of one of the involved organizations.

Due to the exposed nature of this relay server, it is designed to be secure by default. There is only one way of connecting to it, which is the encrypted and authenticated SSH protocol. The protocol transmitted over SSH is designed to be concise and easily audited.

Development is focused on placing as little trust as possible into the relay server. Technical steps are being taken to limit what data can be monitored at the relay server. Some of these features, however, are *not* implemented in the experimental Uplink feature in RCE 10 yet. In all RCE 10.x versions, data transmitted to and from tools is safely encrypted against unauthorized access from *outside* users, through the standard security features provided by the SSH protocol. However, all data could theoretically still be observed by administrators of the relay server. *If this is unacceptable in your setup, please wait for RCE 11 (or later) releases, in which the Uplink feature is planned for non-experimental release. Please note that the feature scope of each release depends on the prioritization of Uplink in relation to project and user needs.*

A typical Uplink setup between two or more organizations involves an Uplink relay server (a specially configured RCE instance) in a location that is accessible from all organization's networks.

This relay server opens an Uplink/SSH port, with one or more sets of credentials for each organization. The recommended approach is to issue one account per person, and potentially additional accounts for technical systems (i.e., servers). This allows tracing activities within the network to individual users or servers. (*Please note that tracing capabilities are rudimentary in the current experimental implementation.*)

Another possible approach is to issue accounts to so-called "gateway" nodes. These are specially configured RCE nodes that make this account usable to other users within the local network without sharing the account's login data itself. End users can then use their individual RCE instances to connect to the gateway node using RCE's local network features. This way, the connection to the relay server can be centrally administered at the gateway while keeping the user's machines simpler.

Please note that in setups based on gateway notes, both the relay server and other organizations can not easily distinguish nodes behind the gateway, and can therefore not easily associate actions with individuals or servers. This may either be an advantage or a disadvantage, depending on your organization's and your partners' IT security policies. Please make sure that the chosen setup is in accordance with the respective IT departments.

In both approaches, any RCE node that connects directly or indirectly (via gateway) to the relay server can use the tools published by the other organizations as compute services, or publish tools themselves to the other organizations. Future versions of the Uplink feature may provide additional features to centrally restrict this ability. If this is a project need for you, please contact us via <rce@dlr.de>.

2.2.1. Security considerations of external tool publishing

As usual in RCE's distribution approach, the binary or script code of "published" tools never leaves the node it is running on. Instead, "publishing" a tool means offering it as a standardized compute service that can be invoked with external input data. The security of tools with regards to potentially unknown input data lies with the tool developer and/or its publisher.

Therefore, it is recommended to avoid publishing tools to external organizations directly from user's machines. Instead, the recommended approach is to only *develop* them on user's machines, and use RCE's *local* publishing options for initial testing with colleagues. Once the tool is sufficiently stabilized, it should then be installed on dedicated server machines, and published to external partners (via Uplink) from there. This allows central administration and security monitoring of these servers.

2.2.2. Example of a Cross-Organization Network

The following figure gives an example of how such a cross-organization network could be structured:

Figure 2.1. Example RCE network

Note

Please note that this diagram makes heavy use of "gateway" nodes, which are not recommended for typical setups at this time. A future version of this guide will contain an updated diagram.

The four project partners in the example all have an internal network of RCE instances which are connected by standard RCE connections. Uplink connections to a relay server are used to connect between the different partners. The relay server is located outside of the organizations networks, and only the relay server has to be reachable via SSH over the internet. Within each organization, users can either connect directly to the relay server (as in the diagram's "Partner D" network, which is typically the preferred approach), or they can use a so-called "gateway" node. Such a gateway node provides a shared SSH/Uplink connection for multiple users. When such a gateway is used, RCE instances in the institution’s internal network can be connected to it by standard (local) RCE connections and still publish tools to the other partners and access tools published by other partners.

Each institution in the example has a different internal setup, all of which are possible:

- Partner A has a dedicated RCE server where the published tools are located, which is connected to the SSH gateway by an RCE connection. All other RCE users in the internal network are connected to this server
- Partner B has put all the tools directly on the SSH gateway instance.
- In Partner C's network, some tools are located on the SSH gateway, but some tools are also published by users directly on their own machines. As long as they are connected to the SSH gateway also those tools can be published to the other partners.
- Partner D has no tool server, instead the users’ computers connect directly to the relay server.

2.3. Notes and Recommendations: RCE's Default Network ("LocalNet") Connections

For default connections, an arbitrary number of incoming network ports can be configured.

> **Note**
>
> TODO This section will be expanded in a future version of this guide; please refer to the User Guide for additional information.

2.4. Notes and Recommendations: SSH/ Uplink Account management

It is **strongly** recommended to use the RCE console's "`keytool uplink-pw`" command to generate secure passwords for Uplink accounts. This command creates a random password with approximately 80 bits of entropy, which is considered secure for accounts based on remote login attempts. Alternatively, you can also use SSH key files -- please refer to the RCE User Guide for this.

In both cases, the login credentials (the password or the SSH key pair) should be generated on the client side, ideally by individual users themselves to minimize the transfer of credential data. The relay server's administrator should only receive either the password's bcrypt hash, or the public part of the SSH key pair.

(Deprecated approach:) When using the built-in administration text mode UI to create SSH accounts, there is an issue when entering passwords including the "@" character. This does not weaken security in any way, but can be confusing for end users when their correctly entered password does not work. It may be advisable to inform users about this.

Hashing SSH passwords with external BCrypt tools has been successfully tested and may be documented in a future version of this guide. Please note, however, that this is only ever useful in automated testing or deployment approaches. For normal user operation, the "`keytool uplink-pw`" console command mentioned above is strongly preferred.

> **Note**
>
> This section is planned to be expanded once the Uplink feature leaves the experimental stage; for example, how to dynamically add SSH/Uplink accounts without a server restart, setting proper Uplink account roles etc.

Chapter 3. Security Properties of RCE Features

This section provides a concise overview of the security properties of various RCE features, especially those related to network activity, security, and information transmission. It is intended to support security reviews and decisions before deploying and using RCE in your organization. For a general overview of RCE's features and operation, please refer to the standard RCE User Guide (available at `https://rcenvironment.de/pages/documentation/documentation.html` for the latest 10.x release) and the RCE Website (`https://rcenvironment.de`).

3.1. Security Properties of RCE's Default ("LocalNet") Network Connections

The default or "internal" network connections of RCE (informally called "LocalNet" connections) are designed for close collaboration within a working group, and are only intended to be used within trusted networks. If you require secure connections across the boundaries of trusted networks (especially across the internet), or collaborate with external partners, the encrypted and authenticated SSH Uplink connections should be used instead.

A default RCE installation does *not* open a network port for these connections unless explicitly configured to do so.

For a general overview of RCE's default network approach, please refer to the standard RCE User Guide (available at `https://rcenvironment.de/pages/documentation/documentation.html` for the latest release).

If RCE is configured to accept incoming network connections, the port number(s) to be used can be freely chosen within the range permitted by the operating system. As RCE should never be run with elevated privileges (SYSTEM, root, or equivalent), this typically limits RCE to ports 1024 and above.

Incoming connections can be restricted to certain IPv4 addresses. However, these filters are static, so this feature is only useful for either restricting connections to localhost for local multi-instance setups, or setting up networks between few, static, and long-living nodes. Both setups are fairly untypical, but may still be of use in certain situations.

3.2. Security Properties of RCE's SSH Server Port

RCE offers a built-in SSH server port, which is disabled by default: An RCE installation does not open a port for accepting SSH connections unless explicitly configured to do so. This port is provided using the Java library Apache SSHD (`https://mina.apache.org/sshd-project/`). We regularily review this and other related libraries for security updates, and issue new RCE releases when necessary.

If RCE's SSH server port is enabled, the port number may be freely chosen.

The accounts used to log into this SSH port are completely independent of system accounts; RCE provides its own account management.

Account passwords are never saved in plaintext. For login verification, only salted BCrypt hashes are stored. SSH key files are supported.

For each SSH account, a single authorization role is selected, which defines which actions are permitted for this account (e.g. workflow monitoring).

These SSH accounts, each with its assigned role and its password hash or SSH public key string, are stored in a JSON file within the instance's "profile" directory The location of this profile directory can be customized.

Accounts can be added or removed by using a provided text mode UI, or by manually inserting or deleting entries in the JSON file.

Connecting to RCE's SSH port does not create or allow any TCP port forwardings; this feature of standard SSH is disabled.

Each RCE instance automatically creates its own SSH server key pair once the SSH port is enabled. The key data is stored in the RCE instance's profile directory.

Client-side strict host key checking is disabled, and a warning is logged when the server-side key has changed. As the SSH login only provides a first line of security, with the actual service security provided by the RCE authorization system, even a successful MITM attack would not have a significant impact. Additionally, without access to the real login credentials, an attacker would have to perform successful MITM attacks on all incoming connections to the relay server, effectively replacing it completely. Such a scenario is highly unlikely, especially because even on success, such an attack would not impact the security of the service authorization system itself (as noted above).

RCE's SSH port supports three modes of operation:

- a custom command shell for RCE administration commands; Unlike standard SSH, this feature *never* provides a direct system shell.
- the deprecated "SSH Remote Access" feature (to be replaced in RCE 11.0);
- the new "SSH Uplink" feature (released as experimental feature in RCE 10.0)

The latter SSH Uplink feature is the recommended mode for offering tools as services to users outside of your organization. Unlike "SSH Remote Access", this mode was specifically designed for this purpose. Further, a special SSH authorization role is provided to restrict SSH accounts to using this mode exclusively. Notably, this also completely disables access to the interactive RCE command shell.

3.3. Security Properties of RCE's Uplink Feature

This protocol was specifically designed to allow different organizations to provide tool execution services to each other. As this naturally involves creating network connections over the boundaries of organization's networks, security is the top priority of its design and implementation. For a general overview of its network approach, please see the "administration" section below.

From an administrator's point of view, the main novelty compared to the older "SSH Remote Access" feature is that it is designed to connect different organizations over a shared "relay" server that can be placed outside the organization's internal network. This eliminates the need to open any incoming network ports in the organization's firewalls. Connections are only established from the internal network to the outside (e.g. the internet), but are never required in the opposite direction.

It is of course also possible to use this feature completely inside the organization's network, for example for securely providing tool execution services between different departments.

Although the current implementation is built on top of the SSH protocol, the actual Uplink protocol is not tied to it. SSH is used as the default transport mechanism as it provides well-tested encryption and login authorization. Technically, this mechanism could be replaced by any other that provides similar features. For example, the Uplink protocol could be expanded/adapted to support TLS connections, which would provide support for CA-based server certificates. At this time, however, there are no specific plans to implement this. Please contact us at `<rce@dlr.de>` if this feature would be relevant for you.

3.4. Security Properties of RCE's Tool Integration

For general information about RCE's Tool Integration concepts and configuration, please refer to the RCE User Guide (available here for the latest 10.x release).

From a security perspective, the service-providing side of Tool Integration consists of a dedicated user (typically an administrator or engineer familiar with the tool) defining a static script that controls the tool's invocation. Typical tasks in this script are custom input pre-processing or conversion, invoking one or more command-line tools, and assembling the final (external) data output.

All inputs and outputs of the service representing the tool is explicitly defined as part of the Tool Integration setup. Inputs and outputs are standardized to data types like Integer, Float, String, File, Directory, Vector, Matrix, or Table.

Beyond input data and (optionally) tool properties, the caller of the tool has no means of influencing the tool's execution.

The tool script is executed by passing it to `bash` on Linux or `cmd` on Windows. (To reiterate: This script is *static*, and always defined by a dedicated user on the *providing* side, *never* by the caller.) However, RCE has no general control over the operations that the administrator/tool integrator performs within that script. Safely handling the data that is received via configured properties or the tool's input data channels lies in the responsibility of the script author.

As part of the tool script editor, RCE offers some support for preventing typical pitfalls in this area. For example, property strings are automatically double-quoted when they are inserted into the editor. Also, certain known-unsafe string contents are disallowed to be inserted into script code; when these are received by the tool, the tool execution is aborted with an error message.

The only data that is implicitly published from a tool's execution is the standard and error output of the executing tools, as well as execution time data. If this is undesired, standard shell/batch features can typically be used to suppress this output.

3.5. Security Properties of RCE's Tool Authorization System

Tools can be assigned to any set of authorization groups, which makes them accessible to any user that is a member of that group.

This assignment is defined on the machine where this tool is "integrated" (ie, where it is defined to be actually run; see the "Tool Integration" section in the RCE User Guide for details).

There is no central authority for group membership. This is an intentional design decision to keep the authorization system flexible and lightweight.

This decentralized approach has, among other things, the benefit of removing administrative overhead when setting up groups between different organizations, for example in the typical use case of a research project with partners from different organizations.

Each group membership is defined by access to a secret 256 bit key. From a user's perspective, this key works like a password required to access a specific group. These group keys are imported into RCE instances as text strings, after which they are available on this instance until they are removed.

The text snippets used to give users access to a certain group should be transported in a secure way, for example via an intranet page with access control.

Users can create any number of authorization groups using their RCE clients.

Due to the decentralized nature of the authorization groups, there is no global password change mechanism, or a central way of revoking group memberships. If this is a concern, periodic group rotation should be used. This can be realized, for example, by using an intranet page with organization-specific access control where the current group access codes are posted.

The group keys that were created or imported into an RCE instance are stored in an encrypted form using the Secure Storage implementation of the Eclipse platform (`https://www.eclipse.org/`). Each user's secure RCE data is encrypted with a random 256 bit master key that is stored in the user's home directory. This way, even if a user accidentally shares his/her whole RCE profile directory to a public location, nobody else can gain access to the stored group keys.

RCE provides console commands for listing all active authorization groups and their assignments, which can be used for automated auditing.