

# The PolicyKit-kde manual

Daniel Nicoletti



# The PolicyKit-kde manual

# Contents

<b>1 Overview</b>	<b>1</b>
<b>2 How it works</b>	<b>2</b>
2.1 Overview . . . . .	2
2.2 The problem . . . . .	2
2.3 The solution . . . . .	2
<b>3 Authorization manager</b>	<b>4</b>
3.1 Manual . . . . .	4
<b>4 Authorization Agent</b>	<b>7</b>
4.1 Manual . . . . .	7
4.2 Authorization Agent dialog . . . . .	7
<b>5 Credits and License</b>	<b>13</b>

### **Abstract**

PolicyKit-kde is a KDE front end to the PolicyKit system that is used to manage authentication.

PolicyKit is a toolkit designed to allow unprivileged processes to speak to privileged processes. It does that by centralizing information of actions and authorized applications.

# Chapter 1

## Overview

PolicyKit-kde is a implementation of PolicyKit tool to the look and feel of KDE.

PolicyKit allows easy and secure password management, it can be used by applications to ask their users for a password. Each application defines a set of actions that can be executed by their program. The application will call PolicyKit to see if the user can perform a given action, if not, the application can issue the auth dialog where the user can enter his/her password, root password, the password of a given group of users or even swipe the finger.

PolicyKit-kde consists of two applications: The Authorization agent that receives requests for authentication, and shows a dialog asking for a password. The Authorization manager that is used to manage the authorizations, it is mainly used by system administrators that may want to change the default behavior of a program policies.

For Qt/KDE developers there is Qt library to allow easy integration with you application and PolicyKit.

For more information of how PolicyKit works, it's design and API visit [PolicyKit Library Reference Manual](#)

## Chapter 2

# How it works

### 2.1 Overview

PolicyKit has a simple way of working, but it requires some design changes from the applications that want to use it to request passwords.

### 2.2 The problem

In GUI applications the common way to gain root privileges is to start it as root, but there are several security risks in doing this method and it does not allow a good actions mapping. There is no way to separate actions like package-install of system-upgrading. All the users who want to use it must have the root password. Another common approach is using sudo but once you start an application with sudo you will have all the rights the root user will have. If for example the GUI application has a dialog to select files that dialog is running as root which means that the user might be able to delete any file on his machine or even copying others user files.

### 2.3 The solution

With PolicyKit this problem is solved. The application in question just need to separate the privileged code into another application, often called helper (which will not have a GUI), then maps the desired actions into a '.policy' file. PolicyKit then loads this file and it can now authenticate applications to use those actions. The use of D-Bus activated applications is the best if not the only, way of putting an helper application to run with root privileges.

With this design the GUI application calls an action of the helper application through D-Bus, which will start the helper with root privileges, and informing

## The PolicyKit-kde manual

it which action was requested and which application has requested it. The helper application now calls the PolicyKit agent to see if that application can do the given task, the helper should report if it could do the requested action. In case the helper saw that the application didn't have enough rights the GUI will then need to ask PolicyKit to obtain an authorization.

When PolicyKit receives the request to obtain an authorization it issues an available Agent, which might happen to be PolicyKit-kde if available. After a successful authentication the GUI application needs to call the helper repeating the same operation again.

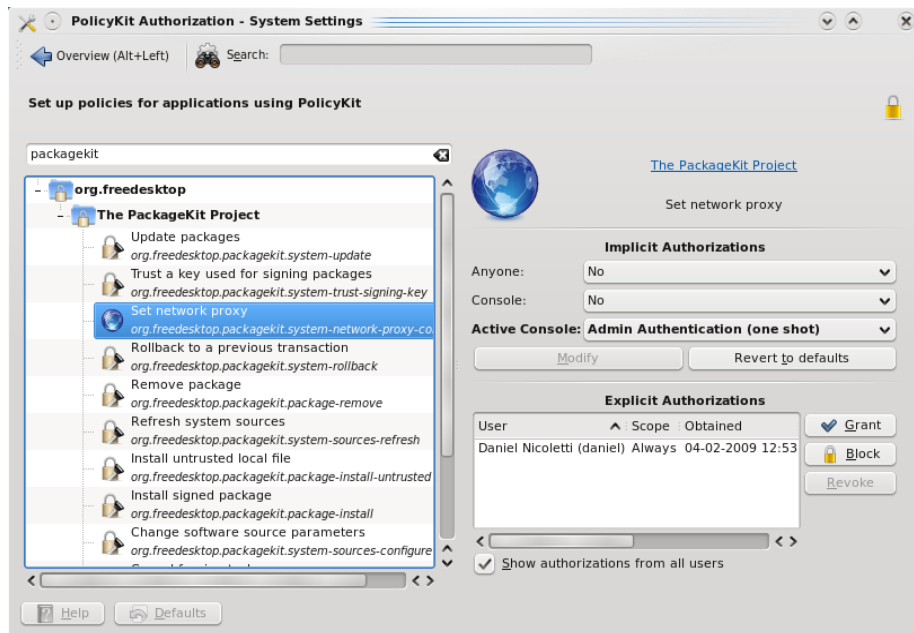
## Chapter 3

# Authorization manager

### 3.1 Manual

The Authorization manager is the application that system administrators can use to easily change the default behavior of any actions. This page does not aim to explain how to create new actions or define new '.policy' files.

The Authorization screen is divided in two parts, at the left we have all the actions that PolicyKit knows, you are able to search the actions using the search bar at the top, and at the right we have the selected action. This screenshot shows the main Authorization screen:



## The PolicyKit-kde manual

When you select an action it's details will be shown at the right side, the action might have an icon, a description and the vendor name. Next in the view we have the 'Implicit Authorizations' and 'Explicit Authorizations'.

The 'Implicit Authorizations' are authorizations automatically given to users based on certain criteria such as if they are on the local console. These authorizations are read from the '.policy' files that the given application defined, they are the defaults settings of the action. These are the valid values

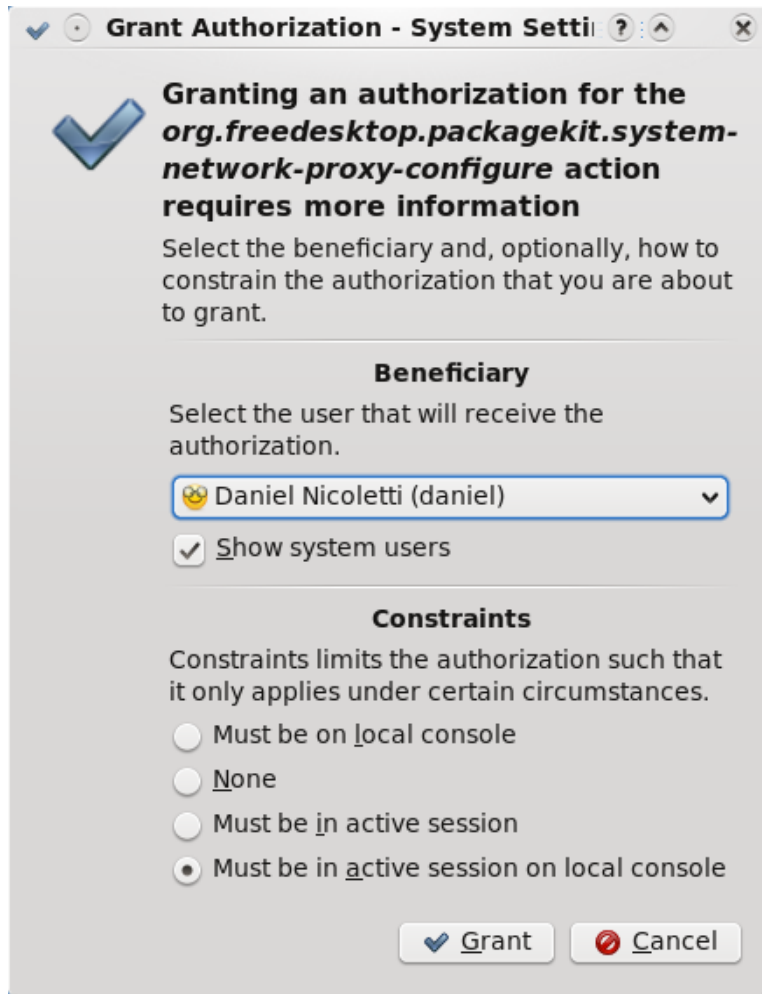
- no
- **auth\_self\_one\_shot**
- auth\_self
- **auth\_self\_keep\_session**
- auth\_self\_keep\_always
- **auth\_admin\_one\_shot**
- auth\_admin
- **auth\_admin\_keep\_session**
- auth\_admin\_keep\_always
- yes

You can change these defaults values simply by changing it on the combo box, the not bold value is the default one so if you want to change one value back you can select it, to make you selection take effect you have to click on the 'Modify' button. The 'Revert to defaults' can be used to change all 'Implicit Authorizations' to it's defaults values. Note that both 'Modify' and 'Revert to defaults' requires you to issue the PolicyKit 'org.freedesktop.policykit.modify-defaults' action which might ask a password.

The 'Explicit Authorizations' are authorizations that are either obtained through authentication process or specifically given to the action in question. The default behavior is to only show the current user explicit authorizations; if you want to see others users explicit authorizations click on the 'Show authorizations from all users', note that this requires you to issue the PolicyKit 'org.freedesktop.policykit.read' action which might ask a password. Blocked authorizations are marked with a 'STOP' sign.

The 'Revoke' button is used to revoke an explicit authorization. Note that this requires you to issue the PolicyKit 'org.freedesktop.policykit.revoke' action which might ask a password.

If you want to specifically grant or block a given user of performing a given action you can click on the 'Grant' or 'Block'. The following screenshot you see the Grant/Block dialog:



To grant/block explicit authorizations you have to select the user that will receive the authorization. You can also select the 'Constraints' to limit the authorization such that it only applies under certain circumstances.

**WARNING**

Be aware that explicit blocking and authorization might self lock you of performing the given action so be sure of what you are doing

Note that this requires you to issue the PolicyKit 'org.freedesktop.policykit.grant' action which might ask a password.

## Chapter 4

# Authorization Agent

### 4.1 Manual

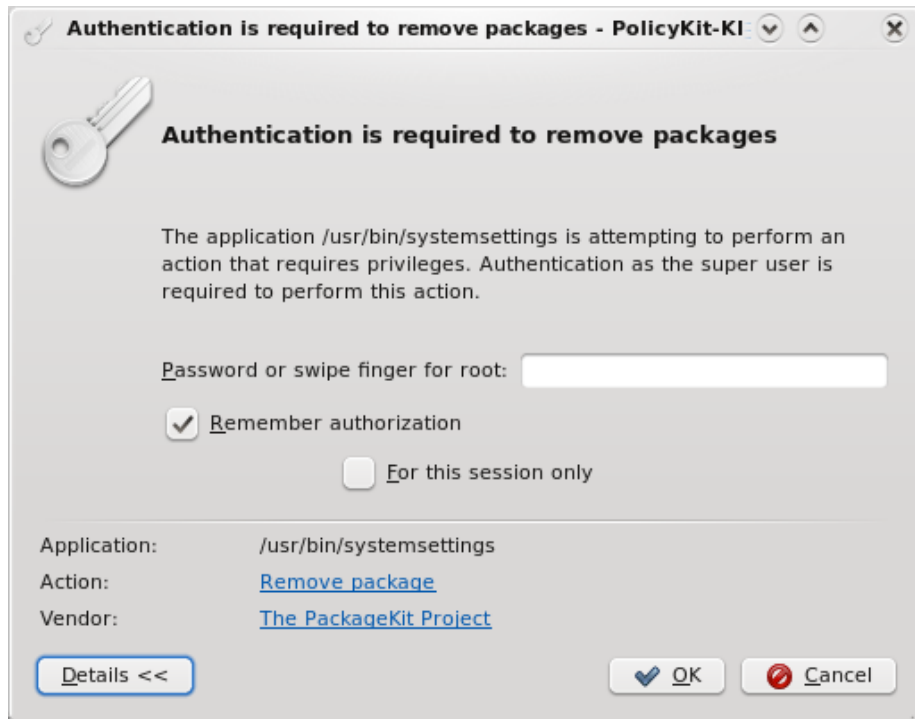
The Authorization Agent is the application that is called whenever an user wants to obtain a given authorization. It's a D-Bus activated daemon which uses 'libpolkit-grant' that in turn uses PAM for authentication services (however, other authentication back-ends can be plugged in as required).

### 4.2 Authorization Agent dialog

The appearance of the authentication dialog depends on the result from PolicyKit and also whether administrator authentication is defined as 'authenticate as the root user' or 'authenticate as one of the users from UNIX group wheel' or however the PolicyKit library is configured (see the PolicyKit.conf(5) manual page for details). Note that some of the screenshots below were made on a system set up to use the [ThinkFinger](#) PAM module. The text shown in the authentication dialogs stems from the PolicyKit .policy XML files residing in /usr/share/PolicyKit/policy and is read by the authentication daemon when an applications asks to obtain an authorization. Thus, what the user sees is not under application control (e.g. it's not passed from the application) which rules out a class of attacks where applications are trying to fool the user into gaining a privilege.

The authentication dialog where the user is asked to authenticate as root using the password or swiping the finger. The details shows the application that's requesting the action, the action itself and the action vendor. If clicking in the action link it will open the authorization manager pointing to the given action, and the vendor might also provide a link for the given action that will be fired when clicking on the 'Vendor' link:

## The PolicyKit-kde manual

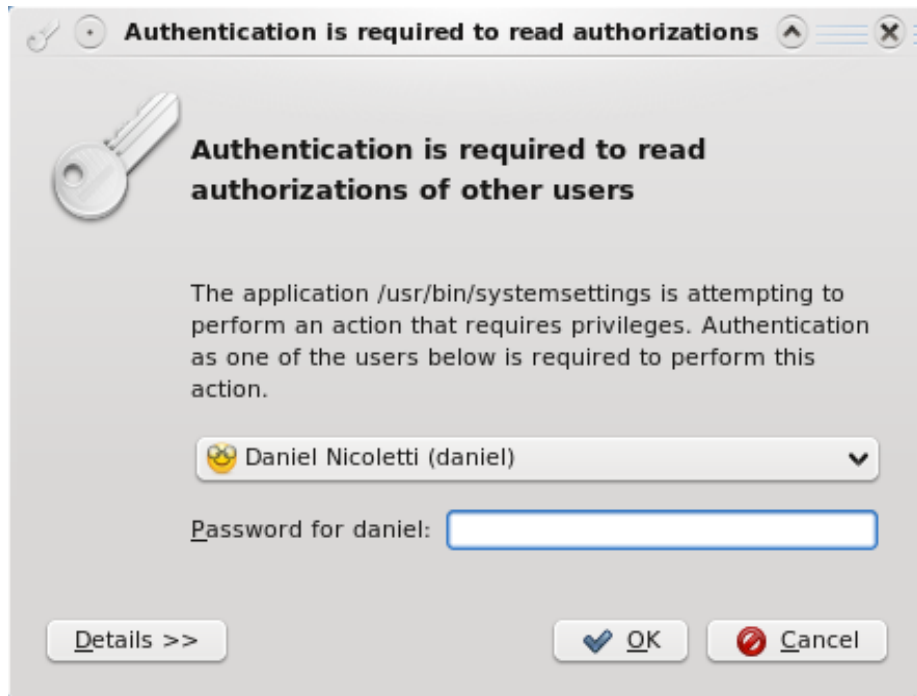


Authentication dialog where the user is asked to authenticate as an administrative user and PolicyKit is configured to use the root password for this:



## The PolicyKit-kde manual

Authentication dialog where the user is asked to authenticate as an administrative user and PolicyKit is configured to use a group for this:



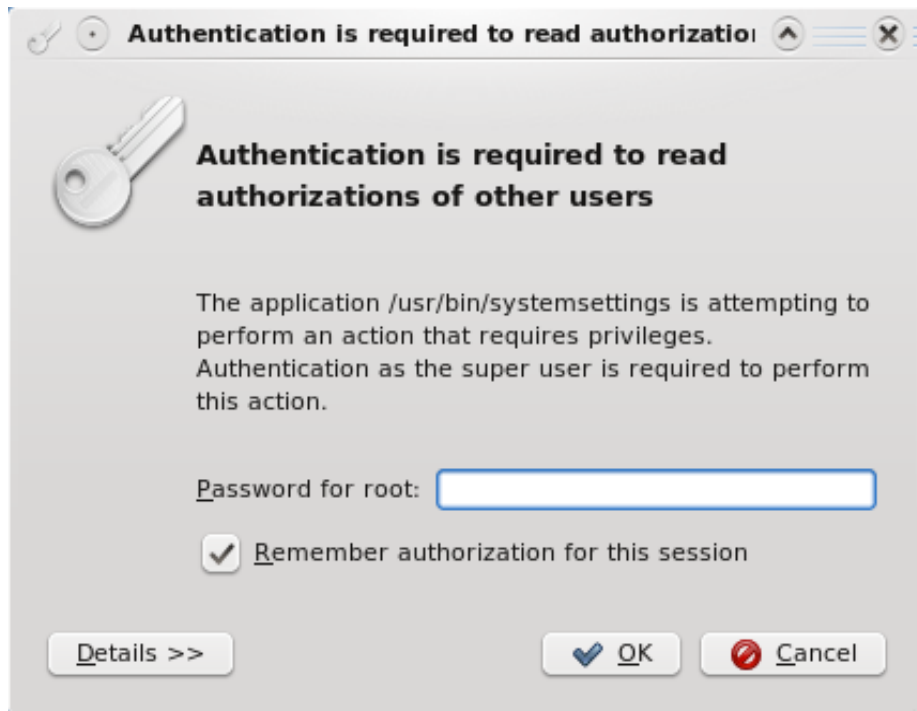
Same authentication dialog, showing drop down box where the user can be selected:



Authentication dialog showing an Action where the privilege can be retained indefinitely:



Authentication dialog showing an Action where the privilege can be retained only for the remainder of the desktop session:



## Chapter 5

# Credits and License

PolicyKit-kde

Program copyright 2008-2009 Daniel Nicoletti

Documentation copyright 2008-2009 Daniel Nicoletti

This documentation is licensed under the terms of the [GNU Free Documentation License](#).