## NAME

polkit − Authorization Framework

## OVERVIEW

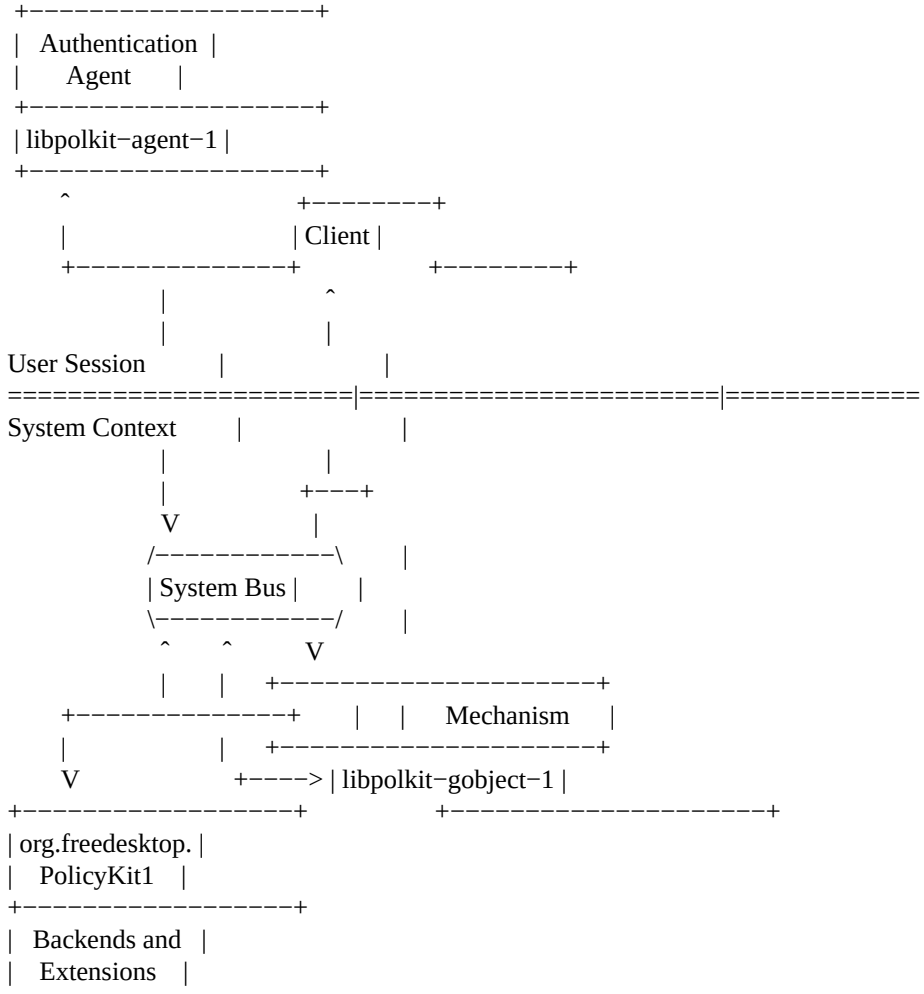
PolicyKit provides an authorization API intended to be used by privileged programs (“MECHANISMS”) offering service to unprivileged programs (“CLIENTS”) through some form of IPC mechanism such as D−Bus or Unix pipes. In this scenario, the mechanism typically treats the client as untrusted. For every request from a client, the mechanism needs to determine if the request is authorized or if it should refuse to service the client. Using the PolicyKit API, a mechanism can offload this decision to a trusted party: The PolicyKit Authority.

In addition to acting as an authority, PolicyKit allows users to obtain temporary authorization through authenticating either an administrative user or the owner of the session the client belongs to. This is useful for scenarios where a mechanism needs to verify that the operator of the system really is the user or really is an administrative user.

## SYSTEM ARCHITECTURE

The system architecture of PolicyKit is comprised of the *Authority* (implemented as a service on the system message bus) and a *Authentication Agent* per user session (provided and started by the user session e.g. GNOME or KDE). Additionally, PolicyKit supports a number of extension points – specifically, vendors and/or sites can write extensions to completely control authorization policy. In a block diagram, the architecture looks like this:

[IMAGE][1]



```
+------------------+
```

For convenience, the libpolkit−gobject−1 library wraps the PolicyKit D−Bus API using GObject. However, a mechanism can also use the D−Bus API or the **pkcheck**(1) command to check authorizations.

The libpolkit−agent−1 library provides an abstraction of the native authentication system, e.g. **pam**(8) and also facilities registration and communication with the PolicyKit D−Bus service.

See the **developer documentation**[2] for more information about using and extending PolicyKit.

See **pklocalauthority**(8) for information about the Local Authority − the default authority implementation shipped with PolicyKit.

## AUTHENTICATION AGENTS

An authentication agent is used to make the user of a session prove that the user of the session really is the user (by authenticating as the user) or an administrative user (by authenticating as a administrator). In order to integrate well with the rest of the user session (e.g. match the look and feel), authentication agents are meant to be provided by the user session that the user uses. For example, an authentication agent may look like this:

[IMAGE][3]

```
+-------------------------------------------------------------+
|               Authenticate                  [X] |
+-------------------------------------------------------------+
|                                     |
| [Icon]  Authentication is required to run ATA SMART    |
|       self tests                    |
|                                     |
|       An application is attempting to perform an    |
|       action that requires privileges. Authentication |
|       as the super user is required to perform this  |
|       action.                       |
|                                     |
|       Password for root: [_____________________]  |
|                                     |
| [V] Details:                        |
| Drive:  ATA INTEL SSDSA2MH08 (045C)             |
| Device: /dev/sda                    |
| Action: org.fd.devicekit.disks.drive−ata−smart−selftest |
| Vendor: The DeviceKit Project               |
|                                     |
|                    [Cancel] [Authenticate] |
+-------------------------------------------------------------+
```

If the system is configured without a *root* account it may allow you to select the administrative user who is authenticating:

[IMAGE][4]

```
+-------------------------------------------------------------+
|               Authenticate                  [X] |
+-------------------------------------------------------------+
|                                     |
| [Icon]  Authentication is required to run ATA SMART    |
|       self tests                    |
|                                     |
|       An application is attempting to perform an    |
```

```
|       action that requires privileges. Authentication |
|       as one of the users below is required to       |
|       perform this action.                           |
|                                                      |
|       [[Face] Patrick Bateman (bateman)       [V]]   |
|                                                      |
|       Password for bateman: [_____________________]  |
|                                                      |
| [V] Details:                                         |
| Drive:  ATA INTEL SSDSA2MH08 (045C)                  |
| Device: /dev/sda                                     |
| Action: org.fd.devicekit.disks.drive-ata-smart-selftest |
| Vendor: The DeviceKit Project                        |
|                                                      |
|                          [Cancel] [Authenticate]     |
+------------------------------------------------------+
```

See **pklocalauthority**(8) on how to set up the local authority implemention for systems without a root account.

Applications that do not run under a desktop environment (for example, if launched from a **ssh**(1) login) may not have have an authentication agent associated with them. Such applications may use the PolkitAgentTextListener type or the **pkttyagent**(1) helper so the user can authenticate using a textual interface.

## DECLARING ACTIONS

A mechanism need to declare a set of “ACTIONS” in order to use PolicyKit. Actions correspond to operations that clients can request the mechanism to carry out and are defined in XML files that the mechanism installs into the /usr/share/polkit-1/actions directory.

PolicyKit actions are namespaced and can only contain the characters [a-z][0-9].- e.g. lower-case ASCII, digits, period and hyphen. Each XML file can contain more than one action but all actions need to be in the same namespace and the file needs to be named after the namespace and have the extension .policy.

The XML file must have the following doctype declaration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policyconfig PUBLIC "-//freedesktop//DTD PolicyKit Policy Configuration 1.0//EN"
"http://www.freedesktop.org/standards/PolicyKit/1.0/policyconfig.dtd">
```

The *policyconfig* element must be present exactly once. Elements that can be used inside *policyconfig* includes:

*vendor*
: The name of the project or vendor that is supplying the actions in the XML document. Optional.

*vendor_url*
: A URL to the project or vendor that is supplying the actions in the XML document. Optional.

*icon_name*
: An icon representing the project or vendor that is supplying the actions in the XML document. The icon name must adhere to the **Freedesktop.org Icon Naming Specification**[5]. Optional.

*action*
: Declares an action. The action name is specified using the id attribute and can only contain the characters [a-z][0-9].- e.g. lower-case ASCII, digits, period and hyphen.

Elements that can be used inside *action* includes:

*description*
: A human readable description of the action, e.g. “Install unsigned software”.

*message*
A human readable message displayed to the user when asking for credentials when authentication is needed, e.g. “Installing unsigned software requires authentication”.

*defaults*
This element is used to specify implicit authorizations for clients.

Elements that can be used inside *defaults* includes:

*allow_any*
Implicit authorizations that apply to any client. Optional.

*allow_inactive*
Implicit authorizations that apply to clients in inactive sessions on local consoles. Optional.

*allow_active*
Implicit authorizations that apply to clients in active sessions on local consoles. Optional.

Each of the *allow_any*, *allow_inactive* and *allow_active* elements can contain the following values:

no
Not authorized.

yes
Authorized.

auth_self
Authentication by the owner of the session that the client originates from is required.

auth_admin
Authentication by an administrative user is required.

auth_self_keep
Like auth_self but the authorization is kept for a brief period.

auth_admin_keep
Like auth_admin but the authorization is kept for a brief period.

*annotate*
Used for annotating an action with a key/value pair. The key is specified using the the key attribute and the value is specified using the value attribute. This element may appear zero or more times. See below for known annotations.

*vendor*
Used for overriding the vendor on a per−action basis. Optional.

*vendor_url*
Used for overriding the vendor URL on a per−action basis. Optional.

*icon_name*
Used for overriding the icon name on a per−action basis. Optional.

For localization, *description* and *message* elements may occur multiple times with different xml:lang attributes.

To list installed PolicyKit actions, use the **pkaction**(1) command.

**Known annotations**

The org.freedesktop.policykit.exec.path annotation is used by the **pkexec** program shipped with PolicyKit − see the **pkexec**(1) man page for details.

The org.freedesktop.policykit.imply annotation (its value is a string containing a space separated list of action identifiers) can be used to define *meta actions*. The way it works is that if a subject is authorized for an action with this annotation, then it is also authorized for any action specified by the annotation. A typical use of this annotation is when defining an UI shell with a single lock button that should unlock multiple

actions from distinct mechanisms.

The org.freedesktop.policykit.owner annotation can be used to define a set of users who can query whether a client is authorized to perform this action. If this annotation is not specified then only root can query whether a client running as a different user is authorized for an action. The value of this annotation is a string containing a space separated list of PolkitIdentity entries, for example "unix−user:42 unix−user:colord". A typical use of this annotation is for a daemon process that runs as a system user rather than root.

## AUTHOR

Written by David Zeuthen <davidz@redhat.com> with a lot of help from many others.

## BUGS

Please send bug reports to either the distribution or the polkit−devel mailing list, see the link **http://lists.freedesktop.org/mailman/listinfo/polkit-devel** on how to subscribe.

## SEE ALSO

**pklocalauthority**(8) **polkitd**(8) **pkaction**(1), **pkcheck**(1), **pkexec**(1), **pkttyagent**(1)

## NOTES

1. /usr/share/gtk-doc/html/polkit-1/polkit-architecture.png
2. developer documentation
   file:///usr/share/gtk-doc/html/polkit-1/index.html
3. /usr/share/gtk-doc/html/polkit-1/polkit-authentication-agent-example.png
4. /usr/share/gtk-doc/html/polkit-1/polkit-authentication-agent-example-wheel.png
5. Freedesktop.org Icon Naming Specification
   http://standards.freedesktop.org/icon-naming-spec/icon-naming-spec-latest.html