polkit

polkit — Authorization Manager

OVERVIEW

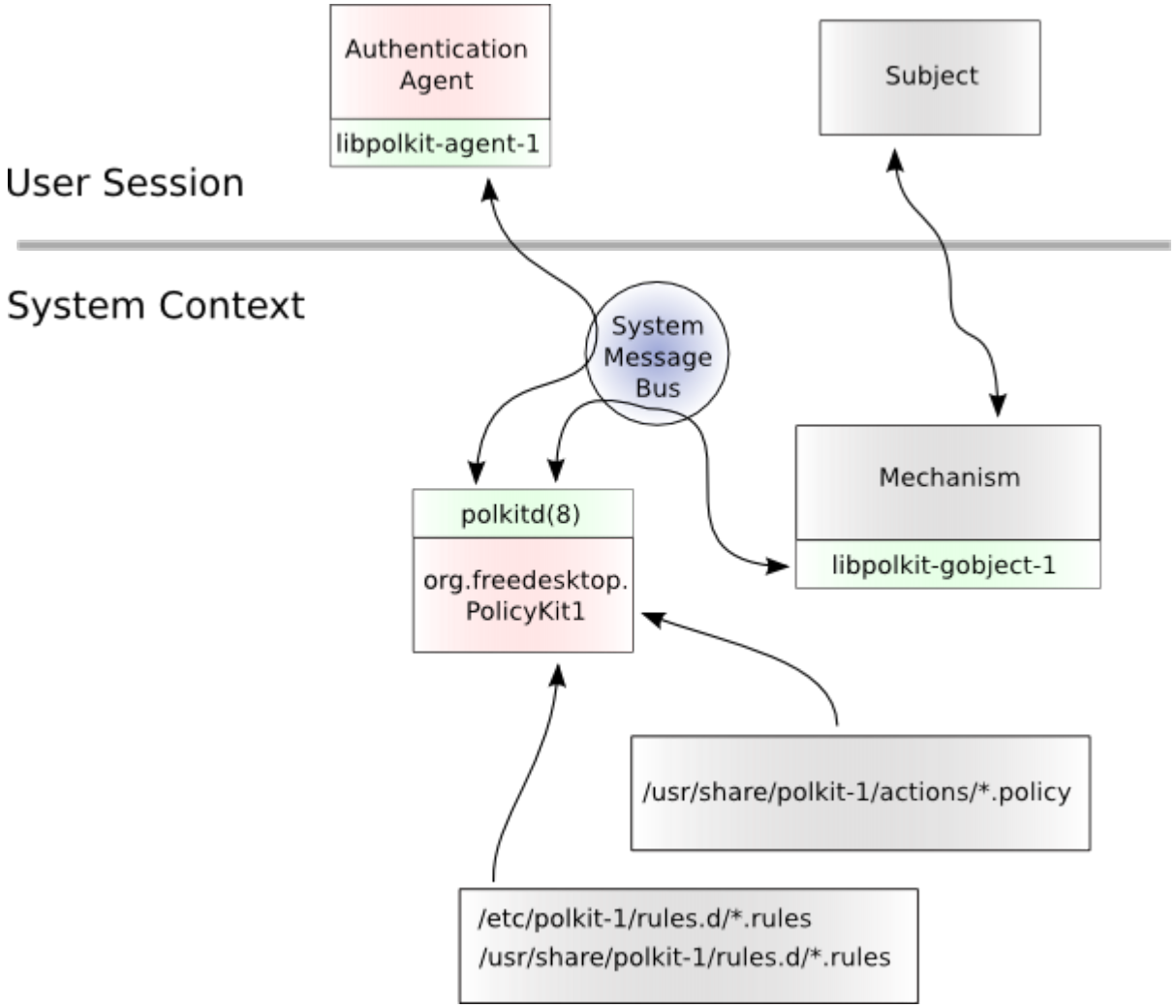
polkit provides an authorization API intended to be used by privileged programs (“MECHANISMS”) offering service to unprivileged programs (“SUBJECTS”) often through some form of inter-process communication mechanism. In this scenario, the mechanism typically treats the subject as untrusted. For every request from a subject, the mechanism needs to determine if the request is authorized or if it should refuse to service the subject. Using the polkit APIs, a mechanism can offload this decision to a trusted party: The polkit authority.

The polkit authority is implemented as an system daemon, polkitd(8), which itself has little privilege as it is running as the *polkitd* system user. Mechanisms, subjects and authentication agents communicate with the authority using the system message bus.

In addition to acting as an authority, polkit 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 polkit is comprised of the *Authority* (implemented as a service on the system message bus) and an *Authentication Agent* per user session (provided and started by the user's graphical environment). *Actions* are defined by applications. Vendors, sites and system administrators can control authorization policy through *Authorization Rules*.



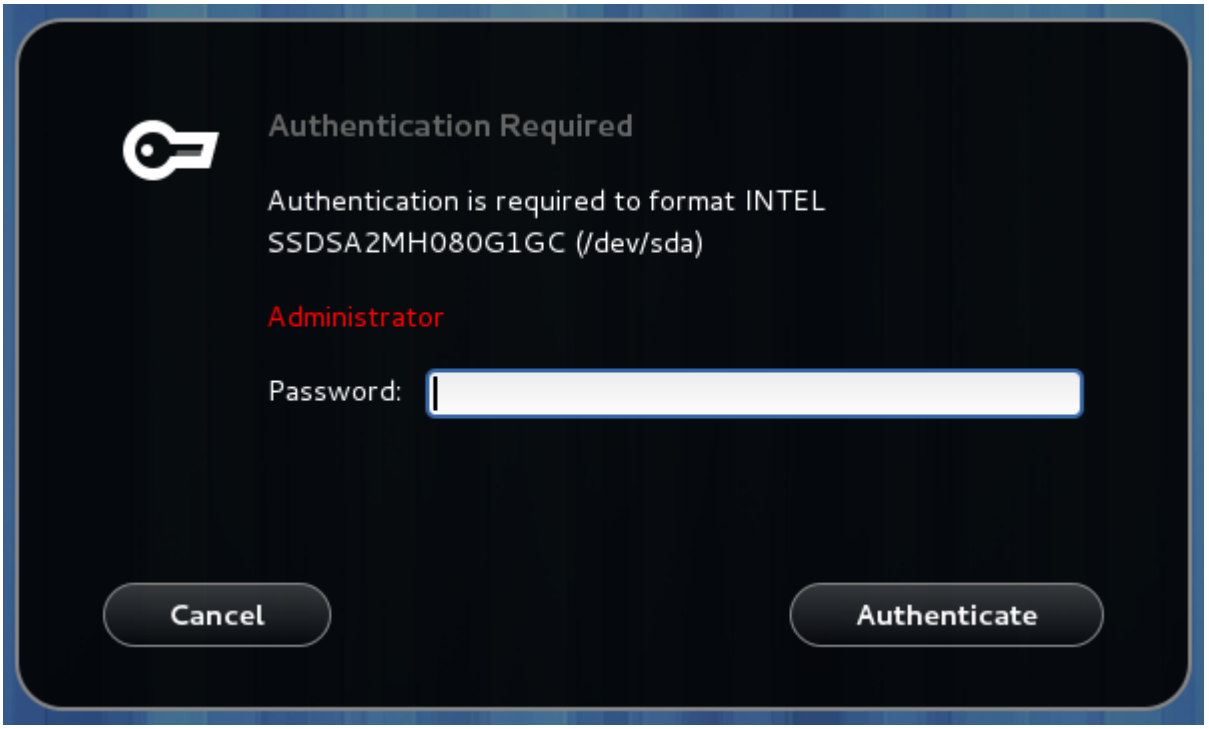
[D]

For convenience, the `libpolkit-gobject-1` library wraps the polkit D-Bus API and is usable from any C/C++ program as well as higher-level languages supporting GObjectIntrospection such as Javascript and Python. 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 polkit D-Bus service.

See the developer documentation for more information about writing polkit applications.

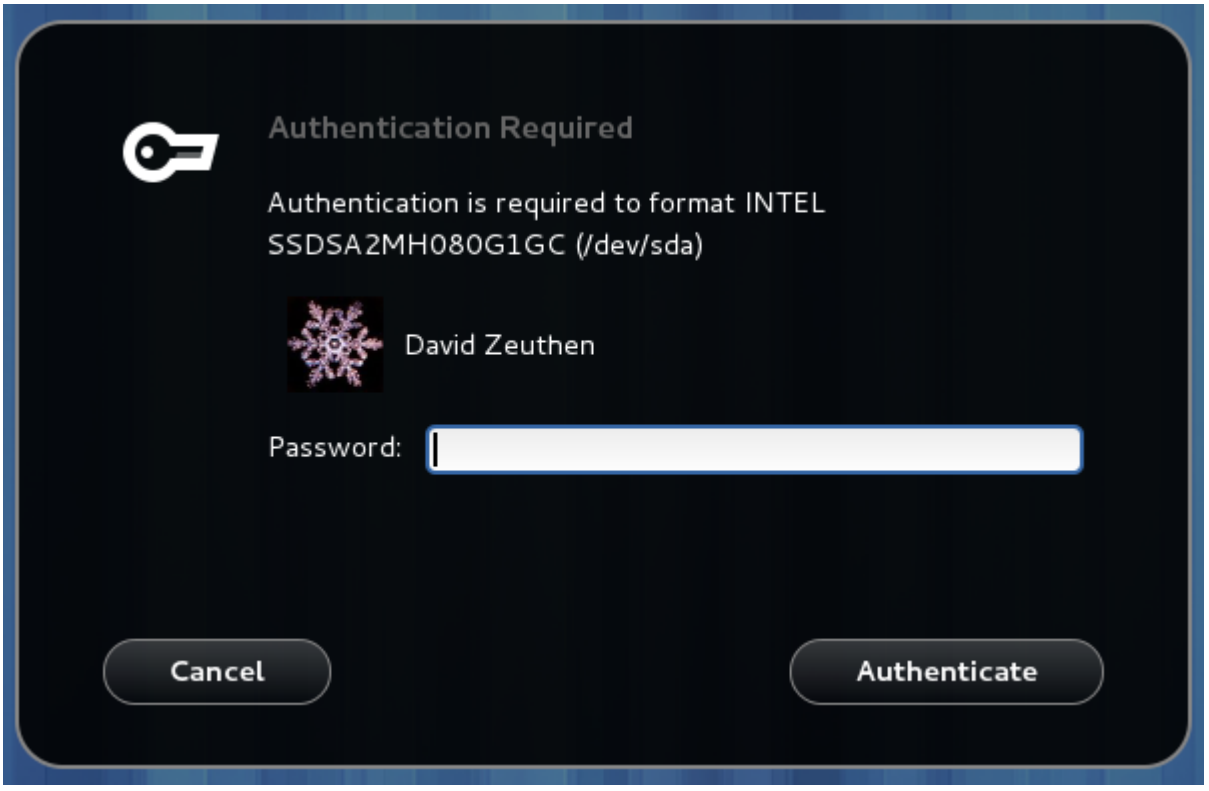
AUTHENTICATION AGENTS

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:



[D]

If the system is configured without a *root* account it may prompt for a specific user designated as the administrative user:



[D]

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 polkit. 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.

polkit actions are namespaced and can only contain the characters `[A-Z][a-z][0-9].-` e.g. 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 polkit Policy Configuration 1.0//EN"
"http://www.freedesktop.org/software/polkit/policyconfig-1.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. Optional.
- *action* — Declares an action. The action name is specified using the `id` attribute and can only contain the characters `[A-Z][a-z][0-9].-` e.g. ASCII, digits, period and hyphen.

Elements that can be used inside *action* include:

- *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* include:
  - *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. Note that this is not restrictive enough for most uses on multi-user systems; `auth_admin*` is generally recommended.

`auth_admin` Authentication by an administrative user is required.

`auth_self_keep` Like `auth_self` but the authorization is kept for a brief period (e.g. five minutes). The warning about `auth_self` above applies likewise.

`auth_admin_keep` Like `auth_admin` but the authorization is kept for a brief period (e.g. five minutes).

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 polkit actions, use the pkaction(1) command.

Known annotations

The `org.freedesktop.policykit.exec.path` annotation is used by the **pkexec** program shipped with polkit - 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.

AUTHORIZATION RULES

polkitd reads `.rules` files from the `/etc/polkit-1/rules.d` and `/usr/share/polkit-1/rules.d` directories by sorting the files in lexical order based on the basename on each file (if there's a tie, files in `/etc` are processed before files in `/usr`). For example, for the following four files, the order is

- `/etc/polkit-1/rules.d/10-auth.rules`
- `/usr/share/polkit-1/rules.d/10-auth.rules`
- `/etc/polkit-1/rules.d/15-auth.rules`
- `/usr/share/polkit-1/rules.d/20-auth.rules`

Both directories are monitored so if a rules file is changed, added or removed, existing rules are purged and all files are read and processed again. Rules files are written in the JavaScript programming language and interface with **polkitd** through the global `polkit` object (of type Polkit).

While the JavaScript interpreter used in particular versions of polkit may support non-standard features (such as the *let* keyword), authorization rules must conform to ECMA-262 edition 5 (in other words, the JavaScript interpreter used may change in future versions of polkit).

Authorization rules are intended for two specific audiences

- System Administrators
- Special-purpose Operating Systems / Environments

and those audiences only. In particular, applications, mechanisms and general-purpose operating systems must never include any authorization rules.

The Polkit type

The following methods are available on the `polkit` object:

```
void addRule( polkit.Result function(action, subject) {...});

void addAdminRule( string[] function(action, subject) {...});

void log( string message);

string spawn( string[] argv);
```

The `addRule()` method is used for adding a function that may be called whenever an authorization

check for `action` and `subject` is performed. Functions are called in the order they have been added until one of the functions returns a value. Hence, to add an authorization rule that is processed before other rules, put it in a file in `/etc/polkit-1/rules.d` with a name that sorts before other rules files, for example `00-early-checks.rules`. Each function should return a value from `polkit.Result`

```
polkit.Result = {
    NO               : "no",
    YES              : "yes",
    AUTH_SELF        : "auth_self",
    AUTH_SELF_KEEP   : "auth_self_keep",
    AUTH_ADMIN       : "auth_admin",
    AUTH_ADMIN_KEEP  : "auth_admin_keep",
    NOT_HANDLED      : null
};
```

corresponding to the values that can be used as defaults. If the function returns `polkit.Result.NOT_HANDLED`, `null`, `undefined` or does not return a value at all, the next user function is tried.

Keep in mind that if `polkit.Result.AUTH_SELF_KEEP` or `polkit.Result.AUTH_ADMIN_KEEP` is returned, authorization checks for the same action identifier and subject will succeed (that is, return `polkit.Result.YES`) for the next brief period (e.g. five minutes) *even* if the variables passed along with the check are different. Therefore, if the result of an authorization rule depend on such variables, it should not use the `"*_KEEP"` constants (if similar functionality is required, the authorization rule can easily implement temporary authorizations using the Date type for timestamps).

The `addAdminRule()` method is used for adding a function may be called whenever administrator authentication is required. The function is used to specify what identies may be used for administrator authentication for the authorization check identified by `action` and `subject`. Functions added are called in the order they have been added until one of the functions returns a value. Each function should return an array of strings where each string is of the form `"unix-group:<group>"`, `"unix-netgroup:<netgroup>"` or `"unix-user:<user>"`. If the function returns `null`, `undefined` or does not return a value at all, the next function is tried.

There is no guarantee that a function registered with `addRule()` or `addAdminRule()` is ever called - for example an early rules file could register a function that always return a value, hence ensuring that functions added later are never called.

If user-provided code takes a long time to execute an exception will be thrown which normally results in the function being terminated (the current limit is 15 seconds). This is used to catch runaway scripts.

The `spawn()` method spawns an external helper identified by the argument vector `argv` and waits for it to terminate. If an error occurs or the helper doesn't exit normally with exit code 0, an exception is thrown. If the helper does not exit within 10 seconds it is killed. Otherwise, the program's *standard output* is returned as a string. The `spawn()` method should be used sparingly as helpers may take a very long or indeterminate amount of time to complete and no other authorization check can be handled while the helper is running. Note that the spawned programs will run as the unprivileged *polkitd* system user.

The `log()` method writes the given `message` to the system logger prefixed with the JavaScript filename and line number. Log entries are emitted using the `LOG_AUTHPRIV` flag meaning that the log entries usually ends up in the file `/var/log/secure`. The `log()` method is usually only used when debugging rules. The Action and Subject types has suitable `toString()` methods defined for easy logging, for example,

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.policykit.exec") {
        polkit.log("action=" + action);
        polkit.log("subject=" + subject);
    }
});
```

will produce the following when the user runs 'pkexec -u bateman bash -i' from a shell:

```
May 24 14:28:50 thinkpad polkitd[32217]: /etc/polkit-1/rules.d/10-test.rules:3: action=[Action
May 24 14:28:50 thinkpad polkitd[32217]: /etc/polkit-1/rules.d/10-test.rules:4: subject=[Subjec
```

The Action type

The `action` parameter passed to user functions is an object with information about the action being checked. It is of type Action and has the following attribute:

string id The action identifier, for example *org.freedesktop.policykit.exec*.

The following methods are available on the Action type:

`string` **lookup**`(` string *key*`);`

The `lookup()` method is used to lookup the polkit variables passed from the mechanism. For example, the pkexec(1) mechanism sets the variable `program` which can be obtained in Javascript using the expression `action.lookup("program")`. If there is no value for the given `key`, then `undefined` is returned.

Consult the documentation for each mechanism for what variables are available for each action.

The Subject type

The `subject` parameter passed to user functions is an object with information about the process being checked. It is of type Subject and has the following attributes

int pid	The process id.
string user	The user name.
string[] groups	Array of groups that `user` user belongs to.
string seat	The seat that the subject is associated with - blank if not on a local seat.
string session	The session that the subject is associated with.
boolean local	Set to `true` only if seat is local.

boolean active Set to `true` only if the session is active.

The following methods are available on the Subject type:

`boolean` **`isInGroup`**`(` string *groupName*`);`

`boolean` **`isInNetGroup`**`(` string *netGroupName*`);`

The `isInGroup()` method can be used to check if the subject is in a given group and `isInNetGroup()` can be used to check if the subject is in a given netgroup.

Authorization Rules Examples

Allow all users in the `admin` group to perform user administration without changing policy for other users:

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.accounts.user-administration" &&
        subject.isInGroup("admin")) {
        return polkit.Result.YES;
    }
});
```

Define administrative users to be the users in the `wheel` group:

```
polkit.addAdminRule(function(action, subject) {
    return ["unix-group:wheel"];
});
```

Forbid users in group `children` to change hostname configuration (that is, any action with an identifier starting with `org.freedesktop.hostname1.`) and allow anyone else to do it after authenticating as themselves:

```
polkit.addRule(function(action, subject) {
    if (action.id.indexOf("org.freedesktop.hostname1.") == 0) {
        if (subject.isInGroup("children")) {
            return polkit.Result.NO;
        } else {
            return polkit.Result.AUTH_SELF_KEEP;
        }
    }
});
```

Run an external helper to determine if the current user may reboot the system:

```
polkit.addRule(function(action, subject) {
    if (action.id.indexOf("org.freedesktop.login1.reboot") == 0) {
        try {
            // user-may-reboot exits with success (exit code 0)
            // only if the passed username is authorized
```

```
            polkit.spawn(["/opt/company/bin/user-may-reboot",
                          subject.user]);
            return polkit.Result.YES;
        } catch (error) {
            // Nope, but do allow admin authentication
            return polkit.Result.AUTH_ADMIN;
        }
    }
});
```

The following example shows how the authorization decision can depend on variables passed by the pkexec(1) mechanism:

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.policykit.exec" &&
        action.lookup("program") == "/usr/bin/cat") {
        return polkit.Result.AUTH_ADMIN;
    }
});
```

The following example shows another use of variables passed from the mechanism. In this case, the mechanism is UDisks which defines a set of actions and variables that is used to match on:

```
// Allow users in group 'engineers' to perform any operation on
// some drives without having to authenticate
//
polkit.addRule(function(action, subject) {
    if (action.id.indexOf("org.freedesktop.udisks2.") == 0 &&
        action.lookup("drive.vendor") == "SEAGATE" &&
        action.lookup("drive.model") == "ST3300657SS" &&
        subject.isInGroup("engineers")) {
            return polkit.Result.YES;
        }
    }
});
```

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

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