Unity8 Policy Kit Documentation

Release 16.04.0

Ted Gould

Mar 15, 2017

Contents

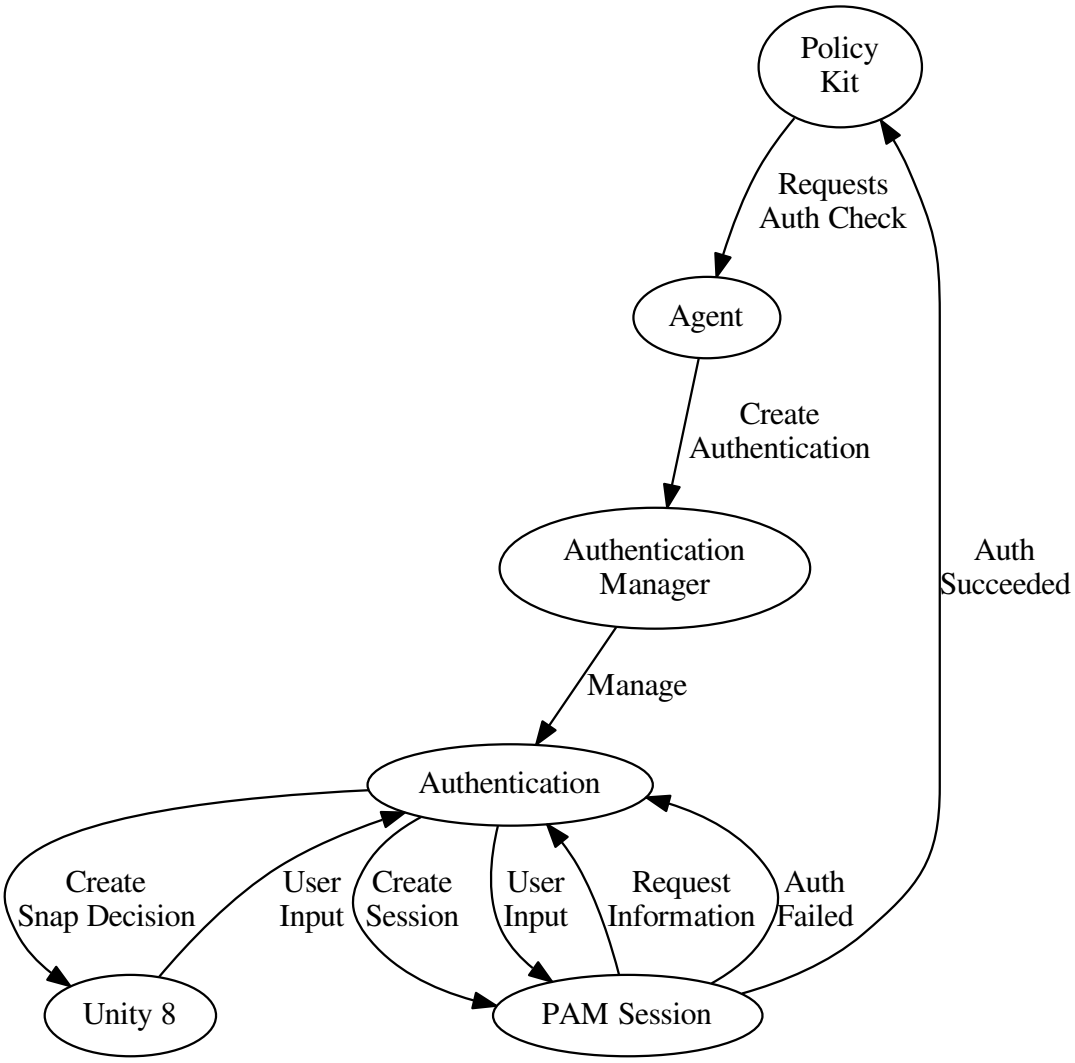
Unity8 PolicyKit project is a PolicyKit agent that is run under the Unity8 desktop. It displays requests from PolicyKit as an Unity8 Snap Decision. Upon receiving a response it then talks to PAM and verifies the information and sends the auth back to PolicyKit.

CHAPTER 1

Architecture

This project dependes heavily on `libpolkit-agent1` which is provided by the PolicyKit project and implements the majority of the DBus interface needed by PolicyKit. It then uses libnotify to communicate with Unit8 and create the snap decision.

Object Interactions



Threads

- **Main Thread**. Starts the process and initializes the base objects then waits for Unix signals for the application.
- **Agent Thread**. Watches the PolicyKit agent interface and requests authentication dialogs based on what PK asks for. It also sends responses back to PolicyKit.
- **Authentication Thread**. Managed by the Authentication Manager this thread handles all authentication dialogs and session management. It ensures that we remain responsive to Unity8's requests.

- **GDBus Threads**. GDBus creates threads that sends messages to the other threads based on which bus the messages are on. (system and session)

CHAPTER 2

Primary Classes

These classes make up the majority of how the code works and interacts, there are a couple of helper classes that are not included but are available in the source code.

Agent

class `Agent`

Class that connects to PolicyKit as the agent and gets events on when PK wants an auth check.

Class to bridge the interface for the PolicyKit agent registration and signaling. It uses a small helper GLib object which is a subclass of libpolicykitagent's PolkitAgentListener class. This is done in GObject, and then this class exists to provide C++ structures around that class.

If requested the agent will use the passed in *AuthManager* instance to create authorization UI's to query the user. It maintains the cancellable objects from GLib and will request the *AuthManager* to cancel any UI requests if PolicyKit asks.

Note The *Agent* Class should be instantiated once, as PolicyKit only allows one agent per session. Creating multiple instances will result in PolicyKit returning an error.

Public Functions

`Agent`(**const** std::shared_ptr<*AuthManager*> &*authmanager*)

`~Agent`()

void **`authRequest`**(**const** std::string &*action_id*, **const** std::string &*message*, **const** std::string &*icon_name*, **const** std::string &*cookie*, **const** std::list<std::string> &*identities*, **const** std::shared_ptr<GCancellable> &*cancellable*, **const** std::function<void(*Authentication*::*State*)> &*callback*)

This is where an auth request comes to us from PolicyKit. Here we handle the cancellables and get a handle from the auth manager for cancelling the authentication.

Parameters

- `action_id`: Type of action from PolicyKit
- `message`: Message to show to the user
- `icon_name`: Icon to show with the notification
- `cookie`: Unique string to track the authentication
- `identities`: Identities that can be used to authenticate this action
- `cancellable`: Object to notify when we need to cancel the authentication
- `callback`: Function to call when the user has completed the authorization

Private Functions

void **unregisterCancellable**(**const** std::string &*handle*)
Disconnect from the g_cancellable

Private Members

std::shared_ptr<*AuthManager*> **_authmanager**
Auth manager used to create authorization UI's

GLib::ContextThread **_thread**
Thread that the agent runs on

std::shared_ptr<AgentGlib> **_glib**
GLib object that interfaces with libpolicykitagent

gpointer **_agentRegistration**
Handle returned by libpolicykit to track our registration

std::map<std::string, std::pair<std::shared_ptr<GCancellable>, gulong>> **cancellables**
All the cancellable objects we're tracking indexed by the cookie that they were associated with.

Private Static Functions

void **cancelStatic**(GCancellable **cancel*, gpointer *user_data*)
Static function to do the cancel

void **cancelCleanup**(gpointer *data*)
Static function to clean up the data needed for cancelling

Authentication Manager

class AuthManager
Class that tracks all the various Authentications that can be in-flight at a given time and gives them a thread to work on.

The authentication manager is mostly a tracker and factory class for *Authentication* objects. When an authentication is requested it creates the object and keeps track of it until it either completes the authentication or is cancelled. It also keeps a GLib mainloop around that the authentication objects can all use.

For bookkeeping purposes this class is also the one that initializes and uninitializes libnotify and makes sure the notification server has the proper capabilities.

Public Functions

AuthManager()

~AuthManager()

std::string **createAuthentication**(**const** std::string &*action_id*, **const** std::string &*message*, **const** std::string &*icon_name*, **const** std::string &*cookie*, **const** std::list<std::string> &*identities*, **const** std::function<void(*Authentication*::*State* > &*finishedCallback*)
Starts an *Authentication*.

Creates the authentication object on the notification thread using the buildAuthentication function. It also creates a more complex callback where, when the callback is called it also removes this authentication from the in_flight map which is tracking *Authentication* objects.

Parameters

- `action_id`: Type of action from PolicyKit
- `message`: Message to show to the user
- `icon_name`: Icon to show with the notification
- `cookie`: Unique string to track the authentication
- `identities`: Identities that can be used to authenticate this action
- `finishedCallback`: Function to call when the user has completed the authorization

bool **cancelAuthentication**(**const** std::string &*handle*)
Cancels an *Authentication* that is currently running.

Parameters

- `handle`: the handle of the *Authentication* object

Protected Functions

std::shared_ptr<*Authentication*> **buildAuthentication**(**const** std::string &*action_id*, **const** std::string &*message*, **const** std::string &*icon_name*, **const** std::string &*cookie*, **const** std::list<std::string> &*identities*, **const** std::function<void(*Authentication*::*State* > &*finishedCallback*)
The actual call to create the object, split out so that it can be replaced in the test suite with a mock.

Private Members

std::map<std::string, std::shared_ptr<*Authentication*>> **in_flight**
All of the *Authentication* objects that currently exist

GLib::ContextThread **thread**
GLib thread for authentications

Authentication

class `Authentication`

Public Types

enum `State`

When the *Authentication* is complete the result of it.

Values:

`CANCELLED`

Authentication was cancelled

`SUCCESS`

Authentication succeeded

Public Functions

`Authentication`(**const** std::string &*in_action_id*, **const** std::string &*in_message*, **const** std::string &*in_icon_name*, **const** std::string &*in_cookie*, **const** std::list<std::string> &*in_identities*, **const** std::function<void(*State*)> &*in_finishedCallback*)

`~Authentication`()

void `start`()

Used to start the session working, split out from the constructor so that we can separate the two in the test suite.

void `cancel`()

Cancel the authentication. Hide the notification if visiable and call the callback.

void `checkResponse`()

Checks the response from the user by looking at the response action and then passes the value to the *Session* object

void `setInfo`(**const** std::string &*info*)

Set the info string to show the user. If there is no info menu item then one is created for the information. If there is currently one it will be updated to be the new string

void `setError`(**const** std::string &*error*)

Set the error string to show the user. If there is no error menu item then one is created for the information. If there is currently one it will be updated to be the new string

void `addRequest`(**const** std::string &*request*, bool *password*)

Add a request for information from the user. This is a menu item in the menu model. If there isn't an item, it is created here, else it is updated to include this request.

Protected Functions

std::shared_ptr<NotifyNotification> `buildNotification`(void)

Build the notification object along with all the hints that are required to be rather complex GVariants.

std::shared_ptr<*Session*> **buildSession** (**const** std::string &*identity*)
Builds a session object from an identity and a cookie. After building it connects to all the signals and passes their calls to the appropriate function on the *Authentication* object.

Parameters

- `identity`: A PK identity string
- `cookie`: An unique identifier for this authentication

void **showNotification** ()
Show a notification to the user, may include building it if it has been built previously.

void **hideNotification** ()
Hide a notification. This includes closing it if open and free'ing the _notification variable. It also will reset the response action and remove all the items from the menu.

void **issueCallback** (*State state*)
Sends the callback, once and only once. It ensures that we don't call it multiple times and that it exits.

Authentication ()
Null constructor for mocking in the test suite

Private Members

std::string **action_id**
Type of action from PolicyKit

std::string **message**
Message to show to the user

std::string **icon_name**
Icon to show with the notification

std::string **cookie**
Unique string to track the authentication

std::list<std::string> **identities**
Identities that can be used to authenticate this action

std::function<void (*State*) > **finishedCallback**
Function to call when the user has completed the authorization

bool **callbackSent** = false
Ensure that we only call the callback once.

guint **actionsExport** = 0
ID returned by GDBus for the export of the action group

guint **menusExport** = 0
ID returned by GDBus for the export of the menus

std::string **dbusPath**
Unique path we built for this authentication object for exporting things on DBus

std::shared_ptr<GDBusConnection> **sessionBus**
Reference to the session bus so we can ensure it lives as long as we do

std::shared_ptr<NotifyNotification> **notification**
If we have a notification shown, this is the reference to it. May be nullptr.

std::shared_ptr<GSimpleActionGroup> **actions**
Action group containing the response action

std::shared_ptr<GMenu> **menus**
The menu model to export to the snap decision. May include info or error items as well as the response item.

std::shared_ptr<*Session*> **session**
The PolicyKit session that asks us for information

Session GLib Interface

class Session
An interface for the session functionality of libpolicykitagent so that we can mock it out.

It is basically impossible to test against PAM without going crazy, so we created an interface to mock it out. This class makes the session interface into nice C++ objects but also virtualizes it so that it can be mocked.

Public Functions

Session(**const** std::string &*identity*, **const** std::string &*cookie*)

~Session()

void **initiate**()
Starts the session so that signals start flowing

void **resetSession**()
Resets the session so it'll start again

core::Signal<**const** std::string&, bool> &**request**()
Gets the request signal so that it can be connected to.

void **requestResponse**(**const** std::string &*response*)
Returns a response from the user to the session.

Parameters

- `response`: Text response

core::Signal<**const** std::string&> &**info**()
Gets the info signal so that it can be connected to.

core::Signal<**const** std::string&> &**error**()
Gets the error signal so that it can be connected to.

core::Signal<bool> &**complete**()
Gets the complete signal so that it can be connected to.

Protected Functions

Session()

Private Members

std::shared_ptr<*Impl*> **impl**
Someone should implement this

class Impl
Implementation of the *Session* class.

This implementation wraps up the PolkitAgentSession object with some static functions that get turned into core::signal's. It also aligns its lifecycle with the GObject one.

Public Functions

Impl(**const** std::string &*in_identity*, **const** std::string &*in_cookie*)

~Impl()

void **requestResponse**(**const** std::string &*response*)
Sends a response to the Polkit *Session*.
Parameters
- `response`: Text response from the user

void **go**()
Internal implementation functions don't have to have good names

void **reset**()
Clears the internal GObject and then reinitializes it to get another session going

Public Members

core::Signal<**const** std::string&, bool> **request**
Signal from the session that requests information from the user. Includes the text to be shown and whether it is a password or not.

core::Signal<**const** std::string&> **info**
Signal from the session that includes info to show to the user

core::Signal<**const** std::string&> **error**
Signal from the session that includes an error to show to the user

core::Signal<bool> **complete**
Signal from the session that says the session is complete, a boolean for whether it was successful or not.

gulong **gsig_request** = 0
GLib signal handle

gulong **gsig_show_info** = 0
GLib signal handle

gulong **gsig_show_error** = 0
GLib signal handle

gulong **gsig_completed** = 0
GLib signal handle

Private Functions

void **clearSession**()
Clears the saved GObject and makes sure to disconnect all of its signals

Private Members

std::string **identity**
Identity we're running against

std::string **cookie**
Cookie of the transaction

PolkitAgentSession ***session**
GObject based session object that we're wrapping

bool **sessionComplete**
A sentinal to say whether complete has been signaled, if not we need to cancel before unref'ing the session.

Private Static Functions

static void **requestCb**(PolkitAgentSession **session*, **const** gchar **text*, gboolean *password*, gpointer *user_data*)
Static callback for the request signal. Passed up to the request C++ signal.

static void **infoCb**(PolkitAgentSession **session*, **const** gchar **text*, gpointer *user_data*)
Static callback for the info signal. Passed up to the info C++ signal.

static void **errorCb**(PolkitAgentSession **session*, **const** gchar **text*, gpointer *user_data*)
Static callback for the error signal. Passed up to the error C++ signal.

static void **completeCb**(PolkitAgentSession **session*, gboolean *success*, gpointer *user_data*)
Static callback for the complete signal. Passed up to the complete C++ signal. Also sets the session complete flag which ensures we don't cancel on destruction.

CHAPTER 3

Quality

Merge Requirements

Submitter Responsibilities

- Ensure the project compiles and the test suite executes without error
- Ensure that non-obvious code has comments explaining it
- If the change works on specific profiles, please include those in the merge description.

Reviewer Responsibilities

- Did the Jenkins build compile? Pass? Run unit tests successfully?
- Are there appropriate tests to cover any new functionality?
- Have the integration tests updated appropriately?
- Can you understand what is happening without asking on IRC?

Manual Integration Test Plan

- **PolicyKit Prompt Check to ensure a prompt comes up when requested**
    1. Install the terminal app
    2. **Execute `pkexec ls`**
        - Ensure that a dialog is shown by Unity8 that asks for your password
        - Make sure that the password is not requested on the terminal
    3. **Enter the user's password in the dialog**

- The command won't show anything if it is successful, error will result in error messages

Index