Displaying and Editing of Permissions of Isolated Applications

Introduction:

Packaging of Applications in Linux:
In Linux, applications are installed in the form of packages, which are a set of files containing the binary/executable of the application, its version, data on its dependencies and any other relevant metadata information. Applications are packaged in different formats. Two common formats are .deb and .rpm, used by Debian-based and Redhat-based distributions respectively.

These packages follow the distribution’s release cycle for upgrades, and have full access to the system’s resources and features. Application developers who wish to run their applications on multiple platforms usually have to package their applications multiple times, for each distribution and packaging format. These packages also have to be maintained individually for each platform.

Flatpaks and Snaps are distribution-agnostic methods of packaging applications. Once an application is packaged in either form, it can be installed on any Linux distribution. They do not follow the distribution’s release cycle, and contain all the dependencies necessary for running the application, separate from the distribution’s native copy of them. Thus, for running an older or newer version of the software than what is provided by the distribution, and for running software not packaged for the distribution at all, Flatpak and Snap versions of an application can be installed.
Sandboxing and Permissions
Unlike natively packaged applications, Flatpaks and Snaps run inside a sandbox, isolated from the system. This restricts them from accessing crucial resources. To remedy this, Snaps use “interfaces” and Flatpaks use “portals” as frameworks for accessing such system resources. Snaps and Flatpaks usually have permissions to access these resources granted by default after installation, but these permissions can be revoked or modified by the user from the command-line or through a 3rd-party app like Flatseal.

Present Situation for Flatpaks:
My task during my Season of KDE 2022 Project was to load and show permissions granted to Flatpaks after installation on Discover’s interface, and to write a System Settings Module for allowing the user to edit these Flatpak permissions. I have worked on showing these permissions on Flatpaks’ application pages, and this work should likely be merged by the end of the SoK.

The task of writing the Settings module to allow users to change these permissions is still unfinished. Thus though Flatpak users will be informed about the permissions a Flatpak needs before the installation, they would not be able to change them.

Present Situation for Snaps:
Snap applications have a button on their application pages to show and enable/disable permissions required by them. This button is visible only after installation, thus Snap users do not have a way of knowing what permissions an app needs before they install it. Further, during the SoK, we decided to not use popups to show Flatpak permissions since they tend to hide permissions away, instead of calling the users’ attention to them. Since clicking the button opens
a popup, Snaps’ way of listing permissions is inconsistent with how Flatpaks do it. Lastly, the permission list lacks icons and any information that would help a non-technical user understand the implications of each permission.

**Problem Statement:**
**For Flatpaks**, build on top of the SoK and write a System Settings module to allow users to edit, revoke and add permissions. **For Snaps**, add more information to each permission item such as icons and brief descriptions, and show them in the application page as opposed to a popup. Also write a System Settings module to allow users to change these permissions.

**Project Goals:**
1. Add more information to make each permission item in Snaps more understandable to users
2. Show permissions required by Snaps on Discover’s application page, and make it consistent with how Flatpaks do it
3. Write a System Settings module for Flatpaks and Snaps each, to allow users to change the permissions granted to their applications
4. Develop an “internal dictionary” that maps a “complex” permission, such as access to a specific bus, with its implication to the user

**Implementation:**
The UI work would be done in QML and the logic in C++, as for all other KCMs.
1. Set up folders called “kcm_fltpakpermissions” and “kcm_snappermissions” as instructed in the [KCM Developer](#)

2. A ListView to show these permissions as a vertical list will be created.

3. The delegate for this ListView would show the following information:
   a. The “name” of the permission
   b. A brief description (like the one that has been implemented in Discover) of what the permission is or asks for
   c. A toggle button, for setting the permission on/off, for simpler permissions such as “Network”.
   d. A textfield (appearing when the user clicks a button called “edit”) or a drop-down menu (for permissions with a fixed range of values, such as “talk”, “own” etc for Session Bus Policy permissions) for more complex permissions

4. A new “SnapPermission” class to be created. Each permission will be instantiated as a “FlatpakPermission” and “SnapPermission” object respectively.

5. All FlatpakPermissions and SnapPermissions will be stored in a data structure such as a QVector or a QMultiMap.

6. A subclass of QAbstractListModel for Snaps and Flatpaks would interface between the data structure and the ListView.

7. org.freedesktop.impl.portal.PermissionStore DBus API will be used using QtDBus Classes to actually set the permission that has been changed, for Flatpaks. For Snaps, this is done through Snapd API.

8. One of the blog posts will document all the new classes/methods/QML components I create.
Timeline:

**Common to all weeks:** blog posts describing the work, challenges faced and facing, holding links to the commits etc.

**Week 1: June 13 - June 20:**
1. Set up the kcm_directories and create the CMakeLists.txt file, as well as the layout.
2. Design the component to use as delegates to show each permission item, it’s icons, buttons, fields, dropdown menus etc., showing the current status of the permission and offering to change it. This is common to Snaps and Flatpaks.
3. Test by using different permissions and ensure that it works and looks acceptable
4. Document the properties and the characteristics of this component for future use

**Week 2: June 21 - June 28:**
1. Ask VDG for feedback on the component after loading some toy permissions for demonstration, and implement their suggestions.
2. Implement the listview and the root item, setting the model to FlatpakPermissionModel and delegate to the component we developed in the previous stage.

**Week 3: June 29 - July 6:**
1. Modify the FlatpakPermissions class and the associated model to conform to the new use, to best fit the delegate. May also shift them to the kcm_flatpakpermissions directory if needed.
2. Instantiate those permission items that are not visible in Discover, such as “wayland”, “x11” etc.
3. Display all the permissions, their icons, descriptions and values in the KCM.
4. Finish any pending issues in the frontend and be done with it.
5. Test by loading permissions for different applications. If there is undesirable behavior for any set or combination of permissions, accommodate those cases.
6. Document all the methods and constructors and their uses in the blog post.

**Weeks 4-5: July 7 - July 21:**

1. Set up the DBus interface to use org.freedesktop.impl.portal.PermissionStore DBus API
   Ensure that we are able to access the functions in the API.
2. Implement the mechanism to actually change the permission as per user’s interaction with the frontend. This is accomplished with the “Set()” function in the DBus API.
3. Implement asking permission using Kirigami.OverlaySheet and reducing or adding list items for complex permissions, as needed.
4. Test this by changing permissions for many different Flatpak permissions, ensuring that they all work as expected.

**Week 6: July 22 - July 29:**

1. Use a QMap to create the internal dictionary to map Flatpak buses with their uses.
2. Show this information in the system settings module, and probably also in Discover’s interface.
3. Document functions associated with this QMap in the post.

**Week 7: July 30 - August 5:**
1. Tie up loose ends in Flatpaks: test the project and fix any bugs, or pending tasks, in either the frontend or the C++ logic.
2. Respond to any feedback that I might get.

**Week 8: August 6 - August 13:**
1. Create SnapPermission class and instantiate all important Snap permissions given in https://snapcraft.io/docs/supported-interfaces.
2. Implement a model to interface between these permissions and the ListView.

**Week 9: August 14 - August 21:**
1. Develop a mechanism to change these permissions, using snapd-qt.

**Week 10 & after: August 22 - September 4:**
1. Tie up loose-ends: respond to feedback, look for bugs, finish up pending tasks.
2. Ask for final suggestions on the work done so far and implement necessary changes.
3. Test all the changes and make it ready to be merged.

**About Me:**
My name is Suhaas Joshi. I am a 2nd-year-student in Christ University, Bangalore, India, pursuing a BTech in Computer Science and Engineering.

I feel I am suited to implement this project because, owing to my participation in Season of KDE, I am already familiar with the project, the codebase, the work done so far, Qt etc.
During SoK, Timothee Ravier and Aleix Pol were my mentors, and we coordinated the work on Matrix and had meetings every Monday.

My native language is Hindi, but I am comfortable with writing and speaking English.

All my previous commits to KDE can be found on this Merge Request:  
https://invent.kde.org/plasma/discover/-/merge_requests/282

IRC NICK: jos22  
MATRIX ID: jsuhaas:matrix.org; DISPLAY NAME: jos22  
EMAIL ADDRESS: joshiesuhaas0@gmail.com