Polkit support in KIO

Name: Chinmoy Ranjan Pradhan
Email: chinmoyrp65.gsoc@gmail.com
IRC Nick: chinmoy

INTRODUCTION

KIO framework is used by KDE applications to provide file management abilities. This framework provides almost all file management functions that one will ever need.

However it doesn't provide any support for file management as a privileged user. So whenever there is a need to perform file management tasks with escalated privileges the application simply shows an error. A casual user may not see it as a problem but for power users, sys-admins, developers this is a huge impedance. Many users get around this by starting the whole application as root. Running Qt/KDE GUI applications as root is not only sub par but also a very dangerous solution. Dangerous because if not handled properly it can cause irreversible damage to the system. To address this issue some KDE applications are disabling being executed as root. For example, the next versions of Dolphin and Kate will show an error message and exit when started with root privileges.

The optimal solution to this problem is to perform only one specific action as root instead of giving root privileges to the whole application. Just here Polkit comes to rescue. Polkit is the standard framework used by Linux desktops to grant privileged actions to unprivileged processes.

The goal of this project is to add Polkit support in KIO. After this addition applications will be able to perform privileged actions after authentication. The post authentication behaviour will be similar to sudo command. This means for sometime the user can perform privileged tasks without having to enter the password repeatedly. However to prevent any accidental damage to the system a warning dialog will be shown prior to performing the task.

DELIVERABLES

1. Polkit support to the following file operations
    - Delete
    - Copy/Cut
    - Trash
    - Rename
    - Creating Symlinks
    - Creating New Files/Folders
    - Changing Ownership and Permissions

2. Integration in dolphin
    - Relaxing assumptions in write protected locations.
    - Emphasising context menu actions and other gui elements of dolphin when location is write protected.
    - Showing necessary warning before executing an action with privilege.
    - Documentation.

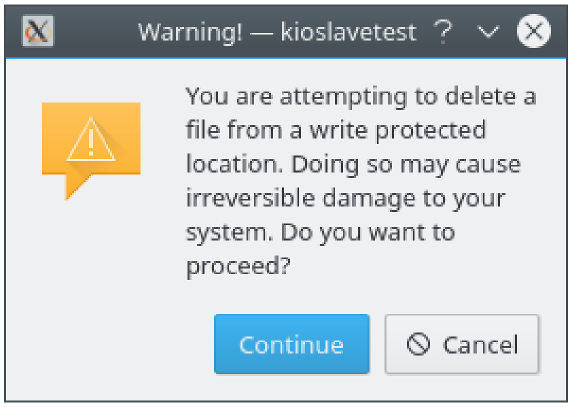
IMPLEMENTATION

I will bring about the aforementioned changes to KIO in following steps:

1. Adding necessary warnings
2. Adding KAuth Helper
3. Refactoring File Ioslave
4. Testing

WARNING DIALOG

A warning is displayed when an application tries to perform a privileged action during the persistence period. It is needed to prevent the application from doing any accidental damage to the system. The warning dialog is basically a KMessageBox and is shown as soon as insufficient privileges are detected. The warning will contain a message describing the cause of warning and the item(s) for which the warning is being shown.



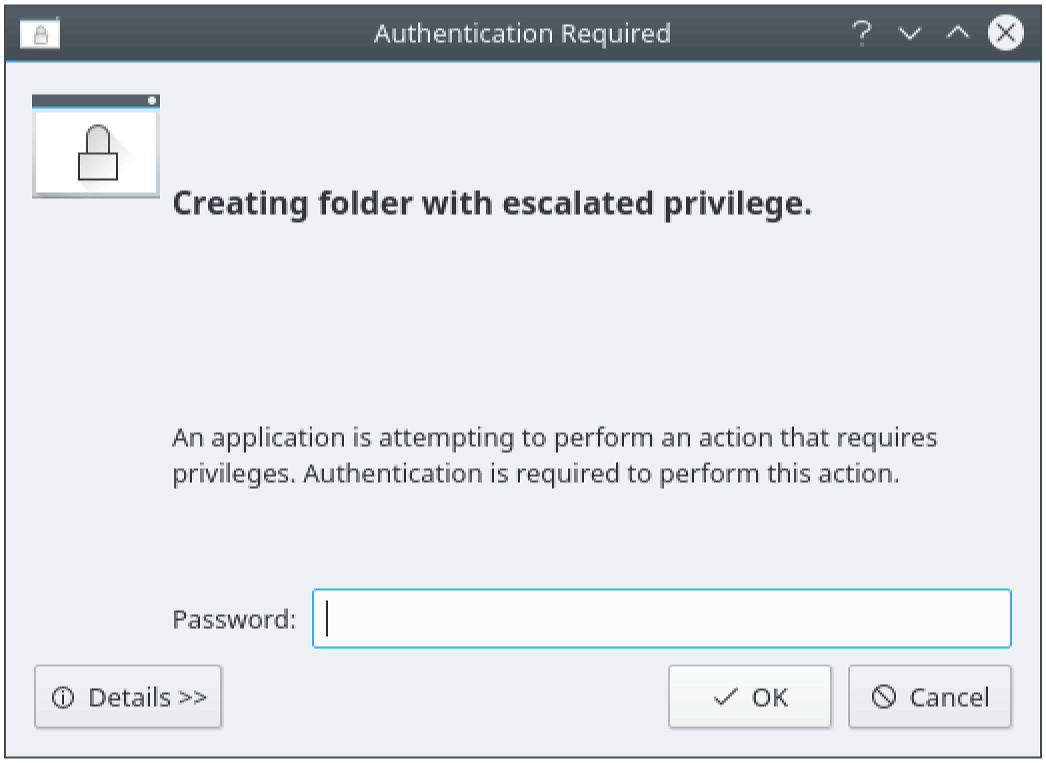
You are attempting to delete a file from a write protected location. Doing so may cause irreversible damage to your system. Do you want to proceed?

❏ *A simple warning*

The warning can either be shown from ioslave or from application through jobuidelegate. Showing it from ioslave provides us with greater control over the contents of warning but we may end up showing similar warning twice. Meanwhile showing warning from the application gives us control over the number of times the warning is to be displayed but on the flip side we will have to check for permissions for every item which in some cases can be time consuming. As we see both the cases have their pros and cons. I will decide upon one of them during the initial period of gsoc.

KAUTH HELPER

KAuth helper is the utility that does the actual work. It first ask the user for his credentials. And after successful authentication it performs the desired action with escalated privilege.



Authentication Required

Creating folder with escalated privilege.

An application is attempting to perform an action that requires privileges. Authentication is required to perform this action.

❏ *KAuth authentication dialog (KDE neon)*

All the file management functions that require privileged environment will be placed in this utility. The kauth actions in the helper utility won't perform the actual file operation. They will only act as meta actions. The purpose of these actions will be to select a suitable (sub)action based on the arguments provided by the ioslave in order to perform the required file operation. The various meta actions, their set of sub actions and the function call each of them will make as a privileged user are listed below.

- Delete Operation

Action	-	del	
Sub-Actions	-	delete_file	[unlink]
		delete_dir	[QDir::rmdir]

- Rename Operation

Action	-	rename	
Sub-Action	-	rename	[rename]

- Copy/Cut/Trash Operation

Action	-	copy	
Sub-Actions	-	open_file	[QFile::open]
		read_file	[read]
		write_file	[write]
		send_file	[sendfile]
		close_file	[QFile::close]

- Creating Files/Folders/Links

Action	-	create	
Sub-Actions	-	put_file	[put]
		create_link	[symlink]
		create_dir	[mkdir]

- Changing Ownership and Permissions

Action	-	change	
Sub-Actions	-	change_own	[chown]
		change_perm	[chmod]

REFACTORING FILE IOSLAVE

File ioslave performs all the file management task. At the moment file ioslave doesn't support performing action as a privileged user. It simply returns an error message when there is insufficient privilege.

To solve this issue I will start by replacing error messages which are displayed when insufficient privilege is detected by a call to method named *execWithRoot*.
This call will be made with the name of meta action, the name of sub action or actual file operation that requires privilege and the path of items upon which the action is to be performed as parameters.
The work of *execWithRoot* is then to prepare a QVariantMap containing the necessary data and call the kauth helper which will perform the desired file operation with escalated privileges.

I have also planned to introduce warning dialog in this part of code but there's also the possibility of the warning being shown from the application's side. It all depends on the relative advantages and disadvantages of both the approaches.
I have a PoC patch which makes use of the former approach.The warning is shown from the ioslave. This patch can be found here [3].

File ioslave has different methods responsible for different file management. I will add polkit support to one method at a time. Then to check if everything works correctly I will add unit test for the corresponding operation. I have listed below the order in which I will add polkit support to the various file management operations.

- Delete Operation

In linux the methods FileProtocol::del and FileProtocol::deleteRecursive combined are responsible for deleting files and directories. The former method deletes one file or one empty directory at a time. Whereas the latter is called when directory is not empty. It then deletes all the items recursively. Currently when deletion of file/directory requires privilege this method shows an error and terminates the whole delete operation.
So to overcome this I will add the usual call to *execWithRoot* which will take care of kauth related stuffs i.e. getting the action, preparing metadata, executing KAuth::Action and if necessary show a warning. The kauth action *del* will be used for doing a privileged delete.

Now it's the delete function where if we choose to show warning from ioslave we might end up showing a similar warning twice which is very unpleasant. And if we choose to show warning from jobuidelegate we might slow down the whole delete operation.

- Rename Operation

It is performed by FileProtocol::rename. It is just a matter of few lines of changes before it can do privileged renaming. The helper action *rename* will be called for this task.

- Copy, Cut and Trash Operation

These operations are performed by FileProtocol::copy because all are just special cases of copy. Performing any one of them with escalated privileges requires them to call the *copy* kauth action.
This action will in turn determine the sub-action (file operation) to be performed, that is to say, file_open, file_read, file_write, file_close or send_file.
As to showing warning dialog choosing between ioslave and jobuidelegate doesn’t really matter. Both the approach will show the warning once and won’t slow down the operation.

- Creating Files, Folders and Links

FileProtocol::put along with FileProtocol::copy are used by KNewFileMenu to create new files. FileProtocol::mkdir is used to create a folder. To create symlinks ioslave uses the FileProtocol::symlink method.
Adding polkit support to these actions should be easy as there’s only one place in these methods where *execWithRoot* needs to be placed. Additionally there will be some changes to KPropertiesDialog because it is responsible for creating new desktop (.desktop) files.
The kauth action *create* will be responsible for performing these operations as privileged user.

- Changing Ownership and Permissions

FileProtocol::chown and FileProtocol::chmod are used for changing ownership and permission respectively. They will make use of the kauth action *change* to change file permissions and ownership with elevated privilege.

TESTING

I will be adding unittest as soon as I finish adding polkit support to an operation just to see if everything is functioning correctly. This will make sure that base is solid and bug free. This way there will be no issues while integrating the patched KIO in dolphin.

To manually test the changes I will make use of kioslavetest which is in KIO's test suite. I have also prepared a small app specifically for this purpose. It's a menu driven app which tries to mimic the file management menu of dolphin. This can be found here [4].

EXPECTED RESULT

At the end of this project I aim for the following results.

- ❏ *Context menu inside a write-protected folder*
- ❏ *Create New menu inside a protected folder*
- ❏ *Authentication for opening a read-protected folder*
- ❏ *Warning dialog (post-authentication)*

POSSIBLE ISSUES

There are couple of issues that may arise during the course of implementation. So I have listed them beforehand.

- As I have mentioned above the warning dialog can be either shown from ioslave or from jobuidelegate. With former method the issue is we might end up with multiple warnings while if we go for latter one we may end up slowing down the whole operation. Of the given options the choice will be made as soon as the program starts. I will also explore other possibilities.

- One of the behavioural changes we want is that once the privilege is escalated for one file operation it should be escalated for other file operations as well. This requires authentication of kauth actions in background. Now polkit has *imply* annotation which can be used for this purpose. However there is no support for annotations in the KAuth library. So there is possibility of some additions to KAuth library.

- Successful integration in dolphin requires emphasising the context menu actions and some of the gui elements of dolphin so as to indicate that the location is write protected but privilege escalation is available. One thing we can do is overlay a lock icon over the existing ones. This way user will know that privilege is required to perform the action. However implementing this idea is somewhat difficult with existing kde frameworks. I will discuss about this with the community soon after finishing the work on KIO. If a better way is found I will consider that instead.

TENTATIVE TIMELINE

Community Bonding Period

During this time I will carry out the following tasks:

- List out all the parts of KIO this project might touch.
- Decide from where should we show the warning dialog, ioslave or jobuidelegate.
- Decide the data and message the warning dialog will display
- Research on polkit's *imply* annotation.
- Find a way to nicely integrate the patched KIO in dolphin.

May

Week 4:

- Start coding the warning dialog.

June

Week 1:

- Start working on kauth helper.
- Work on *del*, *rename* and *change* action.

Week 2:

- Work on *copy* and *create* action. Both are very important actions.
- Add error handling code.
- Comment the new code for future reference.

Week 3:

- Start working on the file ioslave
- Add polkit support to *delete* operation.
- Add polkit support to *rename* operation.

Week 4:

- Add polkit support to *copy/cut/trash* operation.
- Add polkit support to *file/folder creation* operation.

July

Week 1:

- Add polkit support to *changing permission* and *ownership* operation.
- Add unit tests.

Week 2:

- Connect everything i.e. ioslave, helper and warning dialog.
- Manually test all the changes made KIO.
- Discuss with kde users the possible ways to integrate our patched KIO in dolphin.

Week 3:

- Relax the assumptions of dolphin in root owned location.
- Start working on the agreed method of integration.

Week 4:

- Continue working on integration.
- Comment the new code for future reference.
- Add necessary test cases.
- Add documentation.

August

Week 1:

- If integration work is not over then finish it in this week.

Week 2:

- Manually test the integration.
- Make sure our changes to KIO doesn't break other client apps like gwenview, kate, etc.
- Get feedback from users and make necessary changes.

Week 3:

- Buffer time for finishing incomplete parts (if any).

OTHER OBLIGATIONS

I will be completely free from first week of june to last week of july so I can easily give 40+ hrs/week to the project. My college will start from the first week of august. During this time I can only give around 30 hrs/week to the project. However I will work couple of hours extra during rest of the work period to compensate for this time. Except this I don't have any other commitments.

ABOUT ME

I am a 1st year Computer Science student at Dr. Ambedkar Institute of Technology, Bangalore, India. I have been a KDE user for a year and a contributor for about 8 months. I am comfortable with C/C++ and I have working knowledge of bash script and python. I've submitted number of patches which can be found here [1][2]. My contributions are predominantly to KIO so I am very much comfortable with its code base. I have also prepared a PoC patch[3] which adds polkit support to the delete operation. This further strengthened my understanding of the parts of KIO this project involves.

LINKS USED

[1] : https://git.reviewboard.kde.org/users/chinmoyr/ [My submissions]
[2] : https://phabricator.kde.org/diffusion/commit/?authors=chinmoyr [My submissions]
[3] : https://git.reviewboard.kde.org/r/129983/ [PoC Patch]
[4] : https://github.com/crp999/polkitintegrationtest [For Manual Testing]