Introduction
This white paper presents a description of the graphical user interface used by the FBAC-LSM policy manager to interact with users. The policy manager is used by users to construct policies to confine applications using policy abstractions known as functionalities. The policy manager can also load policies into the security module. There are three main windows used to manage policy: the main window which gives an overview of all the confinements and application policies, the application dialog which is used to create or edit application policies, and the learning dialog which can add to policy based on program activity.

The main window
After the program has loaded and the splash screen has been displayed (shown in Figure 1) the main window (Figure 2) shows a list of all the confinements to the left of the dialog. When a user selects a confinement the program advises the user what type of control the user has over the policy and whether the confinement affects the user’s processes.

Figure 1: Policy Manager Splash Screen

1 This splash screen incorporates the photo "Reykjavik" by Robert Whitehead, licensed using Creative Commons Attribution 2.0 Generic.
If the user can alter the policy and it restricts the user’s programs, then the confinement is discretionary and no message is displayed. Otherwise the user is informed which of the following cases is true:

- the confinement restricts the user’s programs but it is mandatory and policy is maintained by another user;
- the confinement does not restrict the user, but the user is authorised to maintain policy which restricts other user(s); or,
- the confinement has no effect on the user and the user cannot alter the policy.

The tabbed view to the right of the confinements list allows the user to see which application policies, functionalities, and settings apply to the selected confinement. The View→Show Advanced Views menu checkbox toggles whether information about applications and functionalities is displayed when they are selected. If advanced views are enabled and an application policy is selected, information about the application is shown which includes the paths of executables and the location of the policy. This view also shows the contained hierarchy of functionalities which the application is assigned as a tree, and the parameter arguments which are passed to functionalities are displayed. As illustrated in Figure 3, by browsing the tree one can view the parameter arguments as they are passed to subsequent functionalities, and these may flow from high level through to low level functionalities. The privileges which are assigned to functionalities are also shown. In this example the Web_Files_Viewer functionality is passed arguments defining a directory containing web files to view and the extensions to use. These details are passed to the File_Viewer functionality and on to the low-level dir_read_access functionality which contains the resulting privileges. Likewise, the advanced view for functionalities shows the hierarchy and privileges assigned to functionalities.
If the user is authorised to maintain policies for the current confinement, buttons are available for managing application policies to:

- add application policies;
- edit application policies;
- remove application policies;
- toggle enforcement mode; or,
- learn rules based on program behaviour.

Additionally users can review and query policies to see what a policy allows an application to do, and to view various other advanced information about an application policy. Adding, editing and reviewing policies are performed via the application dialog. Adding to an application policy using the learning feature is performed via the learning dialog. Both of these dialog windows are described in detail in the following sections. The “Toggle Enforce” button can be used to disable or enable application policies. These features are all also available from the main windows’ menu or the context menu by clicking the right mouse button on the application in the list. The buttons can be hidden using a menu option.

Changes to policy are not made permanent until they are saved to the policy files on disk. This can be initiated, via menu item, keyboard shortcut or when prompted when exiting policy manager. For each policy file, if it already exists it is cleared and the new contents are created and written to disk based on the internal policy representation within policy manager.

The policy manager can also be used to start the FBAC-LSM module and load or reload policy from disk into the module. These features are accessible via the “Security Module” menu. Loading the policy using the policy manager is performed the same way the policy server works.

**The application dialog**

The application policy dialog window is used to create, edit and review application policies. It is made up of a number of pages, which are used to step the user through the process of defining an application policy using a wizard-style interface. Pages are navigated using the “back” and “next” buttons. Page order is non-linear; the order and repetition of pages depends on the options selected. To the left of each page a descriptive textbox describes the purpose of each item on the page. There is a button to hide this textbox if desired.
The application policy being created or edited is stored temporarily and unless the user finishes the process by reaching the last page and clicking “finish” then the process can be cancelled and no changes will be made to the FBAC policy.

In general the page order and which information each page gathers are as follows; although steps are skipped or repeated as necessary:

- **Name**: the name of the application and options for the policy process,
- **Path**: the paths of executables,
- **Base**: the type of interface the application has with the user,
- **Functionalities**: the high level applications which describe the use of the application,
- **Arguments**: application specific information required by functionalities,
- **Privileges**: extra direct privileges,
- **Policy Location**: filename to use when storing policy within a predefined directory, and
- **Review**.

As shown in figure Figure 4, the Name page sets the name of the application profile. The name specified is used when attempting to automate subsequent attempts. The user can choose whether advanced pages and options are shown, and whether the policy manager should attempt to automate steps when possible. For most users the recommended and default setting is not to display advanced options and to automate when possible.

![Figure 4: Policy Manager - Application Dialog - Setting Application Name](image)

The Path page acquires the paths of executables which compose the application. As shown in Figure 5, paths can be entered manually, or chosen with a file dialog. This process can also be automated, in which case the filenames of executables which are stored in directories specified in the $PATH environment variable are searched for matches with the application name. For example, for the application named “opera” the paths `/usr/bin/opera`, and `/usr/bin/X11/opera` are suggested. If no matches are found the user is advised that specifying the name used to start the program from a console facilitates automation of some aspects of policy generation. All automation leads to suggestions which the user is asked to review.
The next policy decision when restricting an application is the way the program interacts with users. Here the user chooses between the available base functionalities. These base functionalities describe the basic needs of various types of applications. In general user-applications are categorised into command line console based applications, or those with graphical user interfaces (GUI). Further distinctions are also made based on whether the application is very simple (just executables) or has configuration settings. As illustrated in Figure 6, at this step in application policy specification the user simply selects the type from a list.

This can also be automated; the executables associated with the application are analysed in a simplistic manner. The programs’ dependencies are examined, either through dynamically linked shared libraries, or if not available for an executable, rpm package management dependencies. If any of these dependencies rely on GUI components such as KDE, QT, GTK, GNOME, or X11, then a GUI base functionality is suggested.

The primary task when creating policy, and the paradigm around which policy is built, is selecting the functionalities which describe what each application is authorised to do. Using the functionality selection page shown in Figure 7, the user can select the appropriate functionalities from the list of those available, and add them
to the list of selected functionalities. There are a number of ways the user can choose to view the available functionalities. By default only high level functionalities are displayed and they are sorted into categories. Categories group similar functionalities. They exist solely to ease the selection of high-level functionalities. Other functionality views can show all functionalities, high level functionalities, or only low level functionalities. Functionalities can be added to the application policy by selecting them and clicking the add button, or by double clicking an available functionality. They can be removed by selecting an added functionality and clicking the remove button.

Figure 7: Policy Manager - Application Dialog - Adding Functionalities

A rudimentary functionality suggestion option has been implemented. The .desktop files which contain information about installed applications are analysed. If a .desktop file corresponds to the application policy, then functionalities are suggested based on iconcategory fields in the file. The iconcategory field is used to sort applications into categories in the menu launch systems of kde and gnome. Some iconcategories map directly to functionalities, for example the .desktop iconcategory “WebBrowser” maps to the FBAC functionality “Web_Browser”. Suggestions can also be made based on program dependencies, which are based on the use of dynamic libraries or rpm dependencies as described previously for the base functionality selection. As shown in Figure 8, when a functionality is suggested it is highlighted in the list of those available and the user is asked to review the suggestions.
The parameter arguments page is shown for each parameter in any of the functionalities which are assigned to the application. Here the user is asked to provide application specific information required by the functionalities assigned. The parameter arguments page is shown in Figure 9. Values can be entered manually either using the text edit box or using parameter type widgets which appear depending on the type of argument value expected. For example, if the parameter is of type “directory” a browse to directory button is available. Likewise files can also be browsed to. Ports and protocols can be selected from a dropdown list. If the parameter is of type “port” then the port number can be specified by its service name (such as http) which is resolved and stored as its numerical value. Currently only IPv4 addresses are supported although extending FBAC-LSM to include other addressing schemes would be trivial. Each parameter has fixed suggested values. Using buttons on the dialog they may be added to the list of values or used unaltered.
The task of specifying parameter arguments is likely the most complicated task in the process of creating application policies. Automating this process lowers the required knowledge of the application needed to create a policy. Supplementing the fixed suggestions for each parameter, the automation implemented can search the filesystem for matching resources. For example the per-user directory can be identified by searching the user home directories for a directory name which contains the name of the application. Opera, for instance, uses the directory `/home/user/.opera`. As shown in Figure 10 the automated process finds the .opera directory and suggests it as the value to use for the parameter. If the option to do so was selected at the start of policy specification, the suggested values are automatically added.

![Policy Manager - Application Dialog – Argument Automation](image)

If the show advanced option, which is disabled by default, was enabled on the first page then the privilege page will be displayed. As shown in Figure 11, this allows the user to add additional low-level privileges directly to the application. This is not recommended for normal users and exists to allow unique requirements to be added or flat (non-hierarchical) application policies to be created for comparison purposes. To add a privilege an operation is selected via the dropdown list, and the resource descriptor values are specified below, then the add privilege button is clicked. Direct privileges to be added can be selected from the list on the right and deleted using the remove privilege button.
If the confinement is configured to store application policies to a directory, the policy location page, shown in Figure 12, specifies which file the policy is stored in within the directory. If all application policies are configured to be stored in a single file this page is not shown. The user can select an existing file from the list to store the policy with other related policies, or a new filename can be specified in the textbox, in which case the file will be created when the policy is saved to disk.

Using the review page the created policy can be reviewed in a number of ways including simple view, privileges, and query tabs. If “View Advanced Review Options” is enabled then another two tabs are also available, the hierarchical view tab and the policy language tab. The simple view tab gives a simple overview of the created policy.

The privileges tab shown in Figure 13 displays a list of all the low-level privileges which are contained in the policy either directly or via functionalities. The privileges shown are the result of parameter argument propagation, where all parameter arguments have been passed to contained functionalities. A filter is provided which allows the privileges to be easily searched and enumerated. For example searching for “file_write” displays all the filenames the application can write to, and searching
“/home/” would show any privileges which specified the home directory in a resource descriptor. It also supports simple wildcards where “*” can represent any sequence of characters. For example “file_read*/home/” shows the privileges which specify the ability to write within the home directory.

The query tab shown in Figure 14, performs a more comprehensive query on the created policy, using the actual access control decision making code to check if the application would be allowed to access a particular resource. Unlike the policy server’s simulation mode, in order to be more assessable this test only considers the policy just created, and does not consider process ancestry or other application policies in other confinements, such as those specified by other users. In practice the application would also be restricted by these factors as well as other security controls on the system such as DAC. The operation describing the type of access, such as “file_write”, is selected from the dropdown list, and the resource descriptors are specified either in the textbox or using the resource descriptor type-specific widgets which appear, such as the “browse for file” button. Once the type of access requested and the resources to access have been specified, pressing the query button will advise whether the action would be “DENIED” or “PERMITTED”. The code used to make this assertion is the same code which is used by the LSM.
The hierarchical view tab shown in Figure 15 displays the details of the policy in a hierarchical tree. The contained functionalities can be browsed, and parameter arguments flowing between the application and contained functionalities can be viewed. This allows the policy to be viewed from high level abstractions and users can drill down to the low level functionalities and the privileges contained within them.

The policy language tab shows the policy as it will be written to disk. This is a representation of the internal state of the construction of the current application policy which has not yet been added to the current confinement.

Once the policy has been reviewed the last page displays a finish button which writes the changes in policy to the main dialog’s internal representation of policy.

**The learning dialog**

The learning dialog can add privileges to an application policy based on what the program tries to do. If a program is attempting to act beyond the privileges granted by the created application policy, the learning dialog can be used to review the access requested and, if appropriate, add privileges to the policy.
The learning dialog has a wizard style interface with a small number of pages, which are accessed in a linear order. As shown in Figure 16, the first page asks the user to choose a learning mode. The options available are enforcing mode or complaining mode. Enforcing mode restricts the program as per usual while in learning mode and all denied access attempts are logged. Complaining mode, on the other hand, allows the program to run unconfined by the current confinement and logs attempts which would have otherwise been denied. The user is reminded that complaining mode can be dangerous, as the program runs unconfined.

The policy manager then sends a command to the FBAC-LSM module to set the enforcing mode for the application policy. A pseudo-random number is sent to the LSM to flag the logs in order to only consider subsequent audit messages. As shown in Figure 17, the user is then prompted to start, and use, the program.
After the user has finished logging access attempts, they click “Next” and the audit logs generated by the FBAC-LSM module are analysed. For each logged attempt the application policy is consulted to check that the privilege is still required – in case privileges have been added that negate the need for the addition. As shown in Figure 18 a page is shown for each logged access attempt which the application policy does not allow. Particularly hazard operations (such as file_write, file_unlink, fs_mount, fs_umount, network_incoming and system_control) are shown in red. If the program tried to execute another program then the operation is shown in orange and the user can select the operation used to execute the program. The values required to satisfy the resource descriptors for the specific requested access are shown in the value list. These values can easily be generalised using the “Wildcard Generalise” button. For example if the program tries to access the image “/test/test.jpg”, then by clicking the generalise button “/test/*.jpg” will be suggested to allow access to all jpg images in the directory. Clicking the “Revert Value” button will undo all changes made to the value.

![Figure 18: Policy Manager - Learn Dialog – Reviewing Access Attempts](image)

After reviewing the need for additional permissions, there are currently two ways the permission can be added to the application profile. Values can be assigned to functionality parameters, or the privilege can be added directly to the application. In the first case the available parameters can be selected from a drop down list. A description of the parameter is shown in a text box. Then the “Add selected value as argument to this parameter” button is clicked. The user then has the option to add the other values and click the “Next” button which appears when using this method. Alternatively clicking the “Add as privilege” button adds a privilege directly to the policy.

After each of the logged attempts has been processed as described above, the user has the option to finalise the changes to the application policy by clicking the finish button. The application policy is returned to enforcing mode.
Conclusion

This white paper has given an overview of the graphical interface of the FBAC-LSM policy manager. FBAC-LSM is available as open source software.