User Data Defence Quick Guide

Hramchenko Vitaliy

November 16, 2012

Contents

Chapter 1

Introduction

User Data Defence is a system based on *SELinux*, which provides protection for your documents when user space applications (such as web browser, PDF viewer) were attacked.

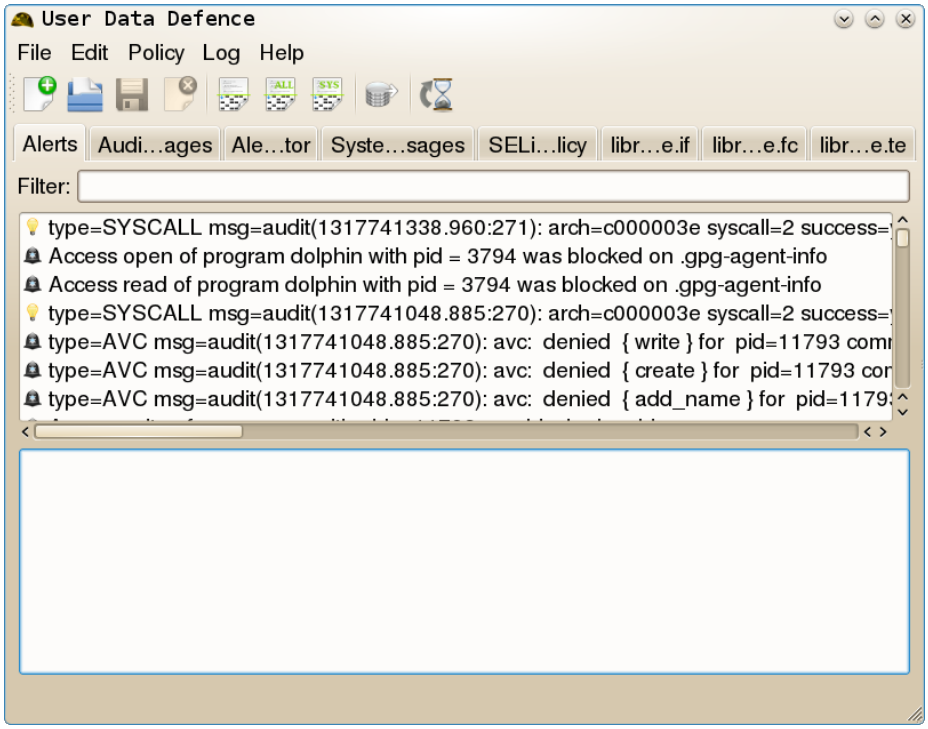

Figure 1.1: Main Window

1.1 How does it works?

User Data Defence includes set of template policies, which makes process of creation *SELinux* specifications for user mode applications simple as never before. Now you could protect documents on your workstation against user mode viruses or program errors.

1.2 Features

This program provides advanced *SELinux* events notification on the Desktop. You could choose notification images and text according to event type or reg exp pattern.

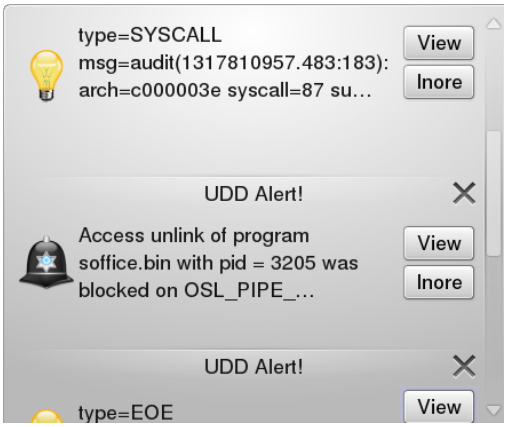

Figure 1.2: Alerts notification

User Data Defence provides you an opportunity to specify a security policy for individual applications, depending on the type of information which need to be processed.

Figure 1.3: UDDTray: Selection mode

One of the main goals for creating User Data Defence was to create replacement of *setroubleshootd* with low CPU usage. Now when system received many alerts in a short period of time CPU usage is not so high.

Chapter 2

Installation

2.1 Requirements

For installing *User Data Defence* on your machine you need to download from system repository some packages:

- SELinux;
- Audit daemon must be installed and activated;
- QT4 development package including D-Bus support;
- audit-libs-devel;
- libselinux-devel;
- dbus-devel.

2.2 Warning

Before installation *User Data Defence* switch your *SELinux* in *Permissive* mode. Type in root console:

`setenforce 0`

Then edit `/etc/selinux/config`: replace `SELINUX=disabled` or `SELINUX=enforcing` with `SELINUX=permissive`.

2.3 Installation

For installation type in your console:

```
git clone git@github.com:Hramchenko/userdatadefence.git
cd ./userdatadefence
qmake (or qmake-qt4 in Fedora)
make
make install
```

For applying changes reboot your computer. If *UDDTray* not started automatically you could start it from console or system menu. At first start you must receive many alerts, otherwise check

that you *SELinux* is in Permissive mode, then check that *UDDaemon* and *UDDBus* are started correctly. If *UDDaemon* was not started you can start it manually from root console:

```
UDDaemon &
```

If you can't find *UDDBus* in process list check contents of file `/etc/audisp/plugins.d/UDDBus.conf` and restart *auditd*.

If *UDDaemon* and *UDDBus* were started check their contexts:

```
ps axZ | grep UDD
system_u:system_r:user_data_defence_bus_t:s0 948 ?  S< 0:00 /usr/sbin/UDDBus
system_u:system_r:user_data_defence_daemon_t:s0 1468 ?  S 0:00 /usr/sbin/UDDaemon
```

If they have another types check that *udd* policy was loaded:

```
semodule -l | grep udd
udd 1.0.0
```

If policy was not loaded install it with `semodule -i udd.pp` command. You may need to customize it for your system.

2.4 Post installation

When your system worked correctly (you don't receive false-positive AVC alerts) you could try to change context of your user. Switch SELinux in *Permissive* mode and type:

```
login -m -s staff_u your_login_name
```

Then append some aliases in ~/.bash_rc profile:

```
alias su="sudo -u sysadm_u -r sysadm_r -t sysadm_t -u root bash"
```

Start new session. You will receive new SELinux alerts. Process it with *UDDTray* generation policy tool. After some days of system usage, when you don't get false-positive alerts, you could try to switch SELinux in *Enforcing* mode (mode with intrusion prevention).

Open graphical console (*konsole*, *gnome-terminal*...) execute `su` or `sudo bash` command. Temporary switch *SELinux* in *Enforcing* mode. You will get new alerts. Append it in a policy.

Warning! Some AVC messages are not shown in alerts list because there are not processed by *auditd* service (such as D-Bus messages). They are shown only in `/var/log/messages` file. You could generate policy for it with *Generate policy for /var/log/messages* option in *UDDTray*.

Enforcing mode will resetting after computer restart. So if your system worked properly in that mode set `SELINUX=enforcing` in `/etc/selinux/config` file.

2.5 Troubleshooting

If your system doesn't work correctly and you don't have any audit messages or a system messages in `/var/log/messages` file try to rebuild SELinux policy with disabled *dontaudit* rules. In root console type `semodule -D`

When problem was resolved you could enable rules with `semodule -B` command.

Chapter 3

Writing SELinux policy

3.1 Creating new policy

This example demonstrates how to create policy for application with graphical interface *Libre Office*.

3.1.1 Creating policy files

For creating new policy select *File->New Policy* in *UDDTray* menu. In *New Policy* window select *Gui application policy*. Then enter existing directory for policy files and its name - `libre_office`.

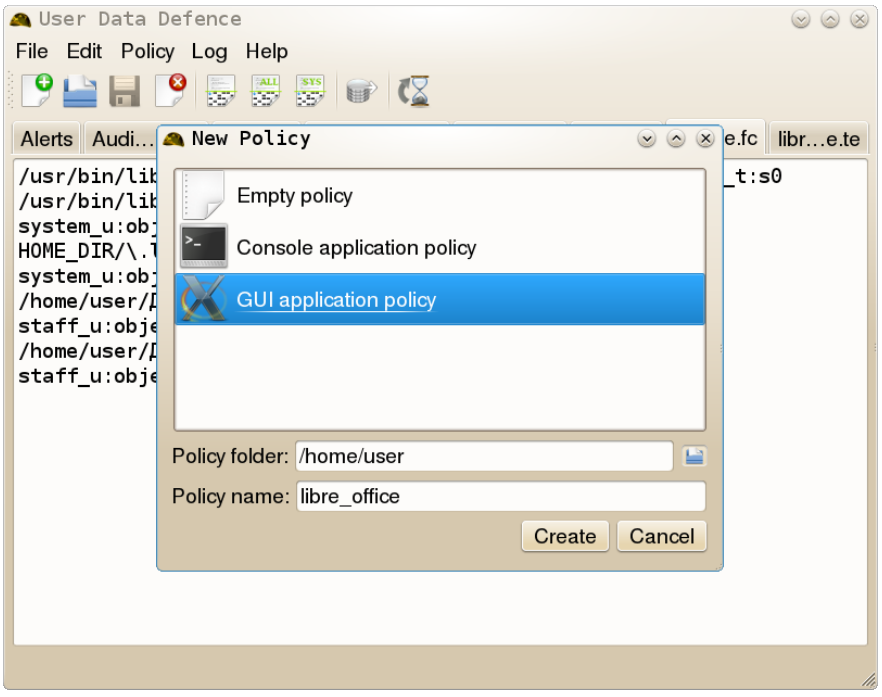

Figure 3.1: Policy creation window

When necessary fields was filled *Create* button will activated. Press it. *UDDTray* generates `libre_office.te`, `libre_office.if`, `libre_office.fc`. Also it creates `udd.te`, `udd.fc`, `udd.if` and `Makefile`, if there are not exists.

3.1.2 Customizing .te file

First line describes policy module name and its version:
`policy_module(libre_office,1.0.0)`
Next four lines contains external types and roles which are used inside a policy:

```
require {
type staff_t;
role staff_r;
}
```

Macro function `udd_gui_application_create(libre_office);` creates:

- `libre_office_t` - domain type;
- `libre_office_exec_t` domain entry point - type of executables which can be used to enter a domain;
- `libre_office_config_dir_t`, `libre_office_config_file_t` - types of files and directories with users configuration files (usually located at home folder).
- `libre_office_tmpfs_t` - type of temporary files.

Line `udd_gui_application_access(libre_office, staff_r, staff_t);` allow user with role `staff_r` and type `staff_t` access to domain `libre_office_t`. It provides domain auto transition from `staff_t` to `libre_office_t`, when user executes file with context `libre_office_exec_t`. Also this instruction allows D-Bus communication between user domain and `libre_office_t`.

Command `udd_gui_application_append_special_domain(libre_office, secret);` creates domain `libre_office_secret_t` with entry point `libre_office_secret_exec_t` and creates types for secret files:

- `libre_office_secret_file_t` secret files type;
- `libre_office_secret_dir_t` directories type. Any file written in that directory by application with domain `libre_office_t` will have context `libre_office_secret_file_t`, any creating folder will have type `libre_office_secret_dir_t`;

Line `udd_gui_application_special_domain_access(libre_office, secret, staff_r, staff_t);` allows user with role `staff_r` and domain `staff_t` access to `libre_office_secret_t`. It creates domain auto transition, and allows their D-Bus communication;

3.1.3 Editing .fc specification

Lets edit `libre_office.fc`.

Change line `/path/to/application -- system_u:object_r:libre_office_exec_t:s0` to `/usr/bin/libreoffice -- system_u:object_r:libre_office_exec_t:s0`.

Copy `libreoffice` to `libreoffice_secret`: `cp /usr/bin/libreoffice /usr/bin/libreoffice_secret`, and replace line `/path/to/application_secret` with `/usr/bin/libreoffice_secret` at the second line.

Next, replace `HOME_DIR/\.libre_office_config(/.*)` with `HOME_DIR/\.libreoffice/`.

Let we have directory with secret files at `/home/secret_documents`, so we need to replace `HOME_DIR/\.libre_office_secret_dir(/.*)?` with `/home/secret_documents(/.*)?` at the last two lines of configuration file.

After all modifications you receive as a result:

```
/usr/bin/libreoffice -- system_u:object_r:libre_office_exec_t:s0
/usr/bin/libreoffice_secret -- system_u:object_r:libre_office_secret_exec_t:s0
HOME_DIR/\.libreoffice(/.*)?  -d system_u:object_r:libre_office_config_dir_t:s0
HOME_DIR/\.libreoffice(/.*)?  -- system_u:object_r:libre_office_config_file_t:s0
/home/secret_documents(/.*)?  -d staff_u:object_r:libre_office_secret_dir_t:s0
/home/secret_documents(/.*)?  -- staff_u:object_r:libre_office_secret_file_t:s0
```

3.1.4 Policy installation

For installation type in your root console:

```
cd /path/to/policy/folder
make
semodule -i libre_office.pp
```

Next step is to change files contexts:

```
restorecon -R /usr/bin/libreoffice* /home/*/.libre_office /home/secret_documents
```

3.2 Generating allow and downtaudit rules

Now we generate `allow` and `donataudit` rules for our *Libre Office* policy (page 6).

Switch your *SELinux* to *Permissive* mode: `setenforce 0`, and run `/usr/bin/libreoffice`. You will receive many alerts in *UDDTray*. Type `libre_office` in a *Filter* field of alerts tab [1]. Select all chosen alerts [2], or type for it *Ctrl+A*. Press *Append to editor* button [3], and activate *Generate policy* button. When *Generate policy* was unchecked you could see allow rules in *SELinux policy* tab [5].

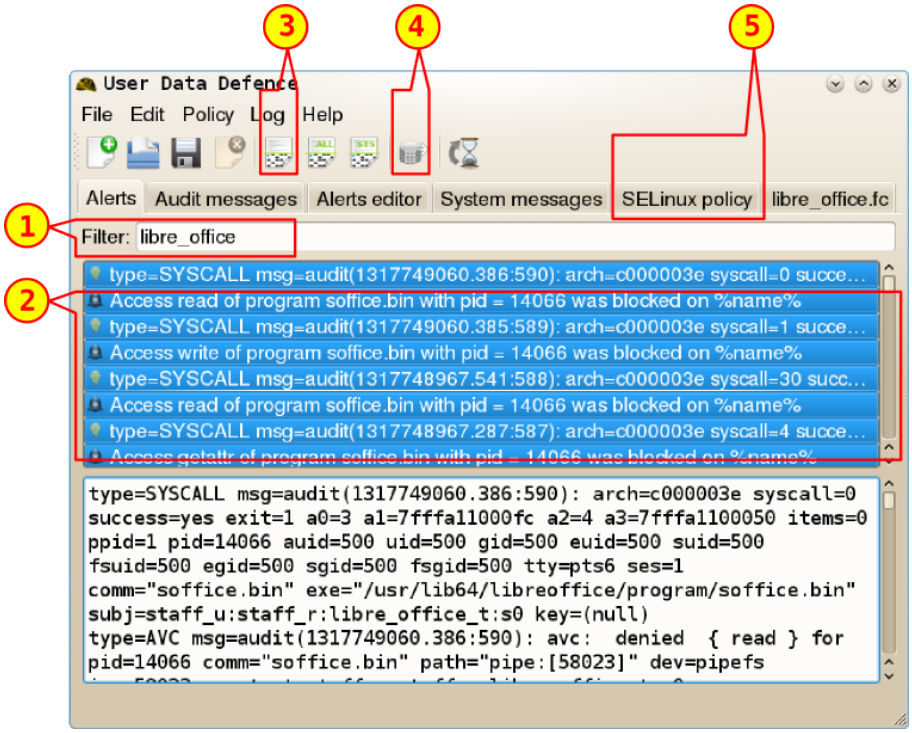

Figure 3.2: Generating allow rules

Append generated rules in `libre_office.te` files. If you don't want to grant some permission for *Libre Office* and want to ignore received messages replace keyword `allow` with `dontaudit`.

Append `allow` and `dontaudit` rules for `libre_office_secret_t` domain (run `/usr/bin/libreoffice_secret`) in the same way. Then rebuild and install policy.

After installation you can try to switch *SELinux* in *Enforcing* mode. Type `setenforce 1` in a root console. When you run *Libre Office* again you will receive new alerts. Append rules for it too.

Warning! Some alerts are not shown at the alerts tab, because there are not system. You could try to generate policy for it using *Generate policy for /var/log/messages* function. Other alerts are not shown because there are `dontaudit` rules for it in other policies. If you want to see alerts for `dontaudit` rules rebuild your policy: `semodule -D`. If you want hide these events type `semodule -DB` in your root console.

3.3 Customizing notification messages

User Data Defence allows you to customize notification of alerts.
Open settings of *UDDTray*: *Edit->Preferences*, and select *Alerts* group.

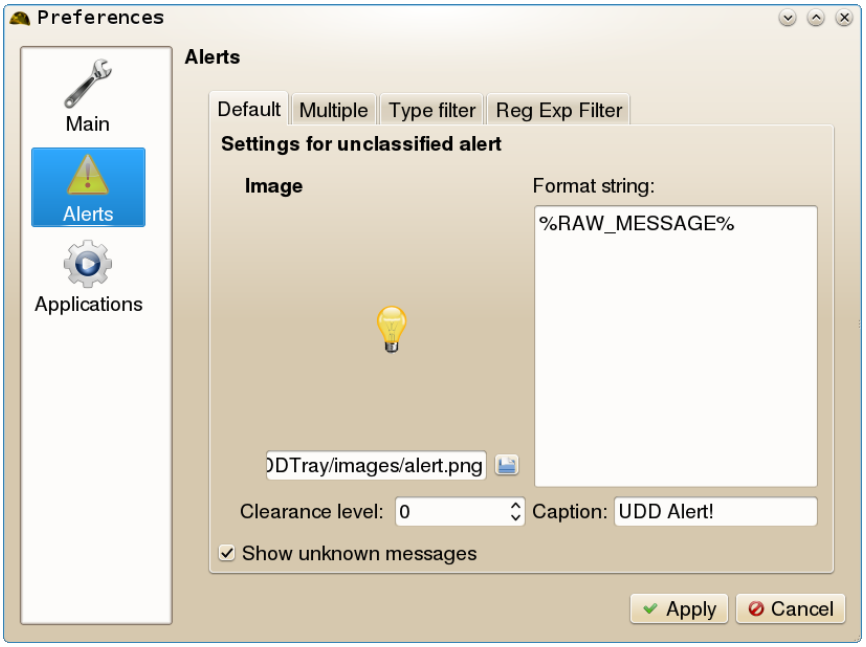

Figure 3.3: Editing alert notification

There are some tabs in this group:

- *Default* - settings for messages, which don't meet any condition, if you disable *Show unknown messages*, this messages are not shown;
- *Multiple* - settings for notifications, which displayed where KNotify is overloaded by *UDDTray* alerts;
- *Type filter* - settings for alerts with certain type;
- *Reg Exp Filter* - settings which meet certain Qt regular expression.

Format string field contains message, which was shown in notification, where `%field%` will be replaced by its value. There are some system values:

- `%RAW_MESSAGE%` will replaced with raw alert text;
- `%alerts_count%` (only in *Multiple* tab) replaced with total count of alerts;
- `%max_level%` (only in *Multiple* tab) replaced with maximum clearance level of received alerts.

Now we create custom notifications for *Libre Office* policy (page 6).

Go to the *Reg Exp Filter* tab. Enter in *Reg Exp* field `type=AVC(.*)libre_office_secret` select alert image and caption, select notification text.

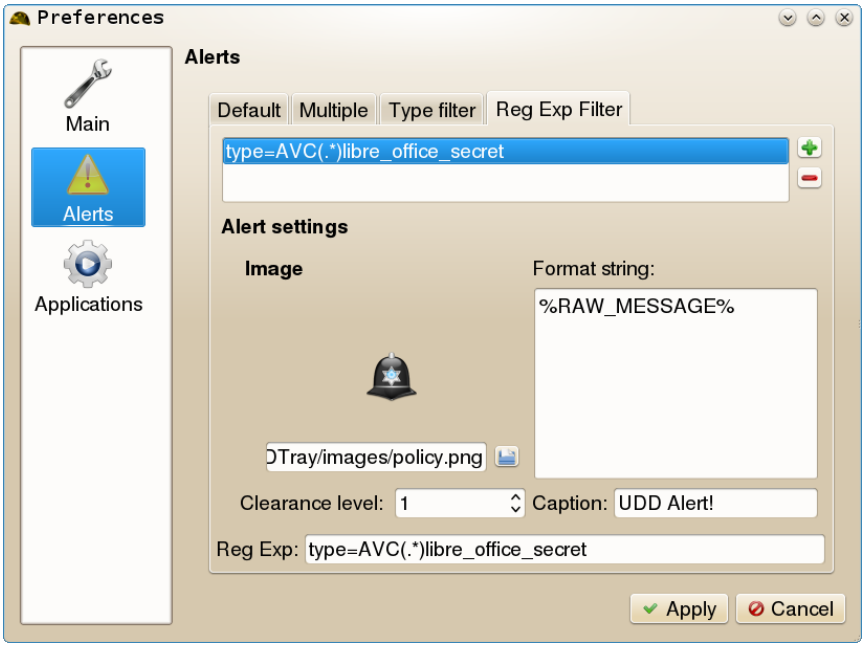

Figure 3.4: Alerts configuration

Now if you receive AVC alerts with `libre_office_secret` string notification system will display something like this:

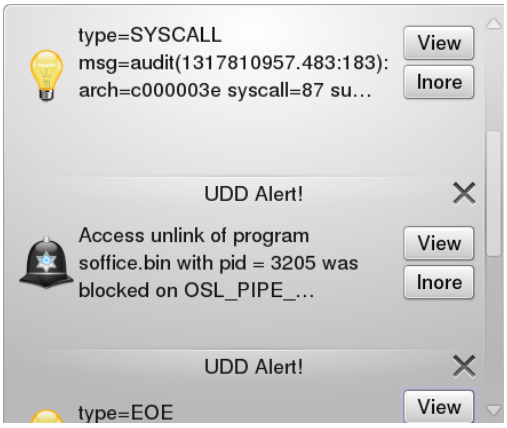

Figure 3.5: Alert notification

3.4 Creating UDDExec profile

UDDExec is an application launcher utility. It executes the calling program in depending on the mode selected in *UDDTray*. Now we configure *UDDTray* to select modes of *Libre Office* policy (page 6).

Open preferences of *UDDTray*: *Edit->Preferences*, and select *Applications* group.

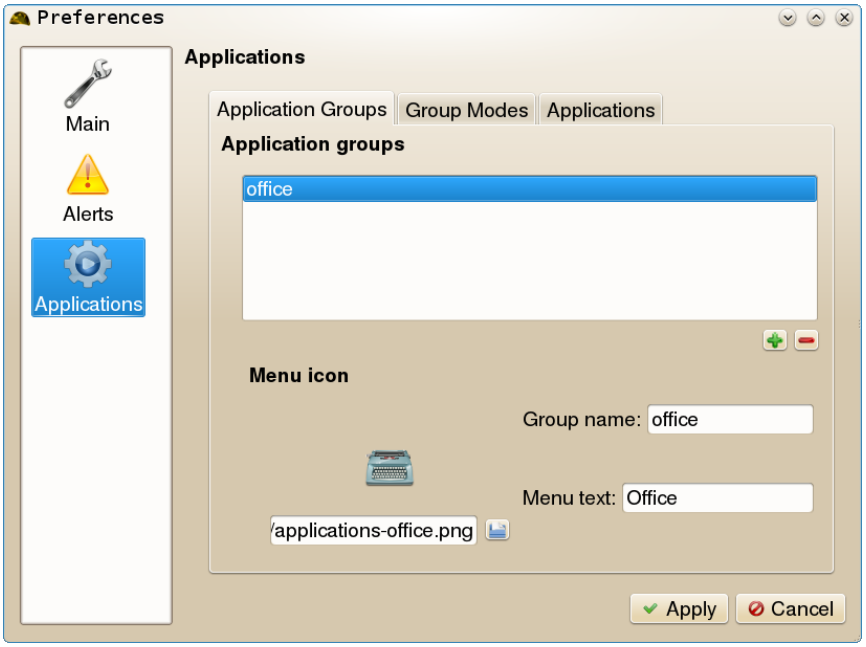

Figure 3.6: Applications groups

Append new group *Office* by activating plus button. Enter group name, and text which will be shown in *UDDTray* menu. Select icon for this group. Go to *Group modes* tab.

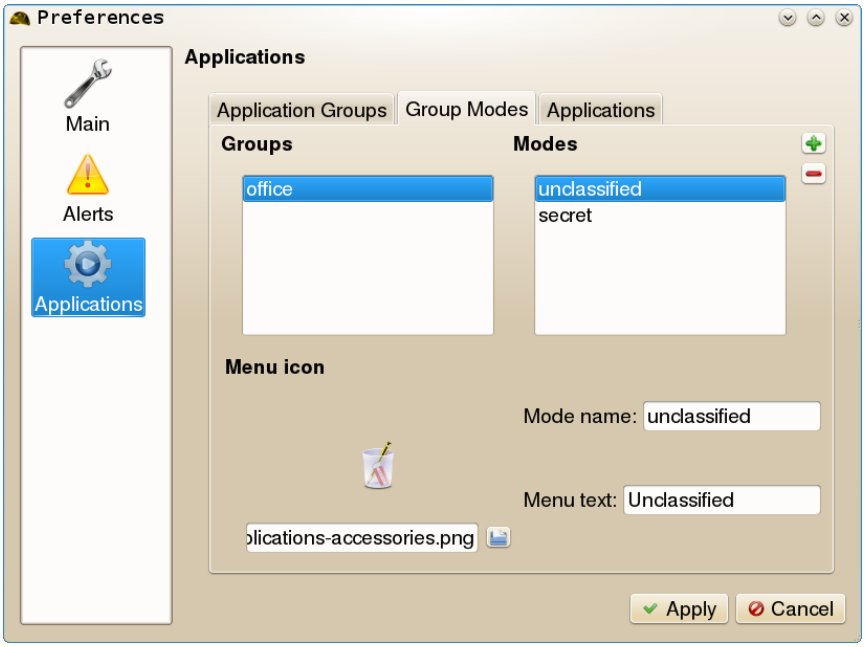

Figure 3.7: Applications groups modes

Append modes *Unclassified* and *Secret* in group *Office*, select its names, menu texts, icons. Open *Applications* tab.

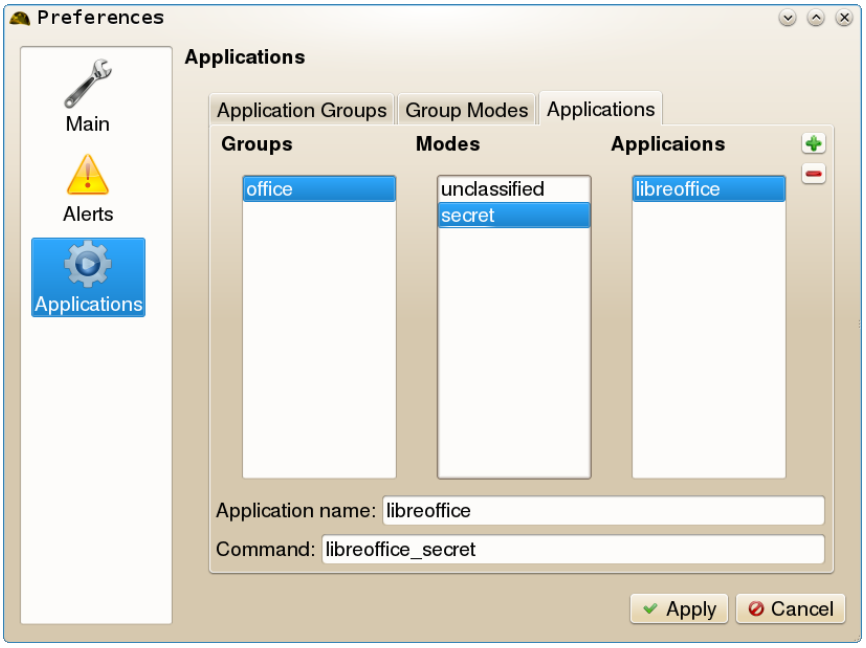

Figure 3.8: Enforced applications

Last step is to append application with name *libreoffice* and command `libreoffice` in mode *Unclassified*, and append application with name *libreoffice* and command `libreoffice_secret` in mode *Secret*.

Now, if you select mode *Unclassified* in *UDDTray* menu, application *libreoffice* will called when you type in users console `UDDExec libreoffice -writer`, and *libreoffice_secret* called on command `UDDExec libreoffice -writer` in *Secret* mode.

Figure 3.9: Selection of UDDExec mode

So if `/usr/bin/libreoffice` has type `libre_office_t`, and `/usr/bin/libreoffice_secret` has type `libre_office_secret_t` you could call program with single command but different domains.

You could replace `.desktop` file of *Libre Office* to your own file with command `UDDExec libreoffice`.

Chapter 4

Architecture

User Data Defence consist of five components: *UDDBus*, *UDDaemon*, *UDDTray*, *UDDExec* and *UDDPolicy*.

4.1 UDDBus

UDDBus is an *auditd* interaction utility. It reads from the input stream of audit events daemon, filters AVC messages from stream and transmits them via D-Bus Service to UDDaemon (this utility is based on code of *sedispatch*, written by D. Walsh).

4.2 UDDaemon

UDDaemon is a daemon of SELinux messages. Daemon receives data from *UDDBus*, accumulates it and provides a data storage. It sends information about new security events to *UDDTray* in a real time.

4.3 UDDTray

UDDTray is a userland component of *User Data Defence*. It is a graphical application which running in the system tray. *UDDTray* performs SELinux alerts through the system notification service *KNotify*. It provides an interface of controlling modes of access control system.

4.4 UDDExec

UDDExec is an application launcher utility. It selects the calling program in depending on the mode selected in *UDDTray*. This utility provide you an opportunity to specify a security policy for individual applications, depending on the type of information which need to be processed.

4.5 UDDPolicy

UDDPolicy is a set of SELinux policy templates for applications with graphical user interface. It provides new macro functions which helps in rapid policy development.