Polkit (Русский)

Ссылки по теме

- **Устранение часто встречающихся неполадок#Разрешения сессии**
- **Sudo**
- **Пользователи и группы**

 **Эта страница нуждается в сопроводителе** 

Статья не гарантирует актуальность информации. Помогите русскоязычному сообществу поддержкой подобных страниц. См. **Команда переводчиков ArchWiki**

Выдержка из **домашней страницы polkit**:

> *polkit — это средство для управления правами приложений пользовательского уровня, позволяющее непривилегированным процессам решать административные задачи: единый интерфейс предоставления прав доступа к привилегированным операциям для непривилегированных приложений при помощи набора правил (политик) и шины D-Bus.*

Polkit используется для управления общесистемными привилегиями. Данный фреймворк используется для предоставления непривилегированным процессам возможность выполнения действий, требующих прав администратора. В отличие от sudo, Polkit не наделяет процесс правами суперпользователя, а позволяет точно контролировать, какие действия разрешены, а какие нет.

Polkit может контролировать отдельные действия, такие как запуск GParted: при этом он проверяет имя пользователя и принадлежность оного к группе, например, является ли он членом группы wheel. Далее Polkit проверяет, какими правами наделены пользователи данной группы (есть ли вообще права на запуск?) и, если всё сходится (пользователь в нужной группе и у группы есть соответствующие права), требует ввести пароль для идентификации пользователя.

Contents

[hide]

Установка

Установите пакет `polkit`.

Агенты аутентификации

Агент аутентификации используется для установления подлинности оператора либо как обычный пользователь (путём ввода пароля пользователя), либо как суперпользователь (путём ввода пароля рута). Пакет **polkit** содержит текстовый агент аутентификации, 'pkttyagent', который используется в качестве запасного варианта.

Если вы пользуетесь графической оболочкой, убедитесь, что установлен графический агент аутентификации и что он **автоматически запускается** при входе в систему.

Такие графические оболочки как **Cinnamon**, **Deepin**, **GNOME**, **GNOME Flashback**, **KDE**, **LXDE**, **LXQt**, **MATE**, и **Xfce** уже имеют в своём составе агенты аутентификации. В других графических окружениях вы можете выбрать одну из реализаций:

- **lxqt-policykit**, который предоставляет `/usr/bin/lxqt-policykit-agent`
- **lxsession**, который предоставляет `/usr/bin/lxpolkit`
- **mate-polkit**, который предоставляет `/usr/lib/mate-polkit/polkit-mate-authentication-agent-1`
- **polkit-efl-git**AUR, который предоставляет `/usr/bin/polkit-efl-authentication-agent-1`
- **polkit-gnome**, который предоставляет `/usr/lib/polkit-gnome/polkit-gnome-authentication-agent-1`
- **polkit-kde-agent**, который предоставляет `/usr/lib/polkit-kde/polkit-kde-authentication-agent-1`
- **ts-polkitagent**AUR, который предоставляет `/usr/lib/ts-polkitagent`
- **xfce-polkit-git**AUR, который предоставляет `/usr/lib/xfce-polkit/xfce-polkit`

Конфигурация

Warning: Do not amend the default permission files of packages, as these may be be overwritten on package upgrades.

Polkit definitions can be divided into two kinds:

- **Actions** are defined in XML `.policy` files located in `/usr/share/polkit-1/actions`. Each action has a set of default permissions attached to it (e.g. you need to identify as an administrator to use the GParted action). The defaults can be overruled but editing the actions files is NOT the correct way.
- **Authorization rules** are defined in JavaScript `.rules` files. They are found in two places: 3rd party packages can use `/usr/share/polkit-1/rules.d` (though few if any do) and `/etc/polkit-1/rules.d` is for local configuration.

Polkit operates on top of the existing permissions systems in Linux – group membership, administrator status – it does not replace them. The .rules files designate a subset of users, refer to one (or more) of the actions specified in the actions files and determine with what restrictions these actions can be taken by that/those user(s). As an example, a rules file could overrule the default requirement for all users to authenticate as an admin when using GParted, determining that some specific user doesn't need to. Or isn't allowed to use GParted at all.

Note: This does not preclude running GParted by means which do not respect polkit, such as the command line. Therefore, polkit should be used to expand access to privileged services for unprivileged users, rather than try to curtail the rights of (semi-)privileged users. For security purposes, **sudoers** is still the way to go.

Actions

Tip: To display Policykit actions in a graphical interface, install the **polkit-explorer**AUR package.

The actions available to you via polkit will depend on the packages you have installed. Some are used in multiple desktop environments *(org.freedesktop.*)*, some are DE-specific *(org.gnome.*)* and some are specific to a single

program *(org.archlinux.pkexec.gparted.policy)*. The command `pkaction` lists all the actions defined in `/usr/share/polkit-1/actions` for quick reference.

To get an idea of what polkit can do, here are a few commonly used groups of actions:

- **systemd-logind** *(org.freedesktop.login1.policy)* actions regulated by polkit include powering off, rebooting, suspending and hibernating the system, including when other users may still be logged in.
- **udisks** *(org.freedesktop.udisks2.policy)* actions regulated by polkit include mounting file systems and unlocking encrypted devices.
- **NetworkManager** *(org.freedesktop.NetworkManager.policy)* actions regulated by polkit include turning on and off the network, wifi or mobile broadband.

Each action is defined in an `<action>` tag in a .policy file.
The `org.archlinux.pkexec.gparted.policy` contains a single action and looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policyconfig PUBLIC
 "-//freedesktop//DTD PolicyKit Policy Configuration 1.0//EN"
 "http://www.freedesktop.org/software/polkit/policyconfig-1.dtd">
<policyconfig>

  <action id="org.archlinux.pkexec.gparted">
    <message>Authentication is required to run the GParted Partition Edito
r</message>
    <icon_name>gparted</icon_name>
    <defaults>
      <allow_any>auth_admin</allow_any>
      <allow_inactive>auth_admin</allow_inactive>
      <allow_active>auth_admin</allow_active>
    </defaults>
    <annotate key="org.freedesktop.policykit.exec.path">/usr/bin/gparted</
annotate>
    <annotate key="org.freedesktop.policykit.exec.allow_gui">true</annotat
e>
  </action>

</policyconfig>
```

The attribute **id** is the actual command sent to **D-Bus**, the **message** tag is the explanation to the user when authentication is required and the **icon_name** is sort of obvious.

The **defaults** tag is where the permissions or lack thereof are located. It contains three settings: **allow_any**, **allow_inactive**, and **allow_active**. Inactive sessions are generally remote sessions (SSH, VNC, etc.) whereas active sessions are logged directly into the machine on a TTY or an X display. allow_any is the setting encompassing both scenarios.

For each of these settings the following options are available:

- *no*: The user is not authorized to carry out the action. There is therefore no need for authentication.
- *yes*: The user is authorized to carry out the action without any authentication.
- *auth_self*: Authentication is required but the user need not be an administrative user.

- *auth_admin*: Authentication as an administrative user is required.
- *auth_self_keep*: The same as auth_self but, like sudo, the authorization lasts a few minutes.
- *auth_admin_keep*: The same as auth_admin but, like sudo, the authorization lasts a few minutes.

These are default setting and unless overruled in later configuration will be valid for all users.

As can be seen from the GParted action, users are required to authenticate as administrators in order to use GParted, regardless of whether the session is active or inactive.

Правила авторизации

Authorization rules that overrule the default settings are laid out in a set of directories as described above. For all purposes relating to personal configuration of a single system, only `/etc/polkit-1/rules.d` should be used.

The `addRule()` method is used for adding a function that may be called whenever an authorization check for action and subject is performed. Functions are called in the order they have been added until one of the functions returns a value. Hence, to add an authorization rule that is processed before other rules, put it in a file in `/etc/polkit-1/rules.d` with a name that sorts before other rules files, for example `00-early-checks.rules`.

The layout of the .rules files is fairly self-explanatory:

```
/* Allow users in admin group to run GParted without authentication */
polkit.addRule(function(action, subject) {
    if (action.id == "org.archlinux.pkexec.gparted" &&
        subject.isInGroup("admin")) {
        return polkit.Result.YES;
    }
});
```

Inside the function, we check for the specified action ID *(org.archlinux.pkexec.gparted)* and for the user's group *(admin)*, then return a value "yes".

Administrator identities

The `addAdminRule()` method is used for adding a function that may be called whenever administrator authentication is required. The function is used to specify what identities may be used for administrator authentication for the authorization check identified by action and subject. Functions added are called in the order they have been added until one of the functions returns a value.

The default configuration for administrator identities is contained in the file `50-default.rules` so any changes to that configuration should be made by copying the file to, say, `40-default.rules` and editing that file.

```
/etc/polkit-1/rules.d/50-default.rules

polkit.addAdminRule(function(action, subject) {
    return ["unix-group:wheel"];
});
```

The only part to edit (once copied) is the return array of the function: as whom should a user authenticate when asked to authenticate as an administrative user? If she herself is a member of the group designated as admins, she only need enter her own password. If some other user, e.g. root, is the only admin identity, she would need to enter in root's password. The format of the user identification is the same as the one used in designating authorities.

The Arch default is to make all members of the group **wheel** administrators. A rule like below will have polkit ask for the root password instead of the users password for Admin authentication.

```
/etc/polkit-1/rules.d/49-rootpw_global.rules

/* Always authenticate Admins by prompting for the root
 * password, similar to the rootpw option in sudo
 */
polkit.addAdminRule(function(action, subject) {
    return ["unix-user:root"];
});
```

Примеры

Debugging/logging

The following rule logs detailed information about any requested access.

```
/etc/polkit-1/rules.d/00-log-access.rules

polkit.addRule(function(action, subject) {
    polkit.log("action=" + action);
    polkit.log("subject=" + subject);
});
```

Disable suspend and hibernate

The following rule disables suspend and hibernate for all users.

```
/etc/polkit-1/rules.d/10-disable-suspend.rules

polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.login1.suspend" ||
        action.id == "org.freedesktop.login1.suspend-multiple-sessions" ||
        action.id == "org.freedesktop.login1.hibernate" ||
        action.id == "org.freedesktop.login1.hibernate-multiple-sessions")
    {
        return polkit.Result.NO;
    }
});
```

Bypass password prompt

To achieve something similar to the **sudo** `NOPASSWD` option and get authorized solely based on **user/group** identity, you can create custom rules in `/etc/polkit-1/rules.d/`. This allows you to override password authentication either **only for specific actions** or **globally**. See **[1]** for an example rule set.

Globally

Create the following file as root:

```
/etc/polkit-1/rules.d/49-nopasswd_global.rules

/* Allow members of the wheel group to execute any actions
 * without password authentication, similar to "sudo NOPASSWD:"
 */
polkit.addRule(function(action, subject) {
    if (subject.isInGroup("wheel")) {
        return polkit.Result.YES;
    }
});
```

Replace `wheel` by any group of your preference.

This will result in automatic authentication for **any** action requiring admin rights via Polkit. As such, be careful with the group you choose to give such rights to.

For specific actions

Create the following file as root:

```
/etc/polkit-1/rules.d/49-nopasswd_limited.rules

/* Allow members of the wheel group to execute the defined actions
 * without password authentication, similar to "sudo NOPASSWD:"
 */
polkit.addRule(function(action, subject) {
    if ((action.id == "org.archlinux.pkexec.gparted" ||
     action.id == "org.libvirt.unix.manage") &&
        subject.isInGroup("wheel"))
    {
        return polkit.Result.YES;
    }
});
```

The `action.id`s selected here are just (working) examples for GParted and **Libvirt**, but you can replace them by any other of your liking as long as they exist (custom made or supplied by a package), and so can you define any group instead of `wheel`.

The `||` operator is used to delimit actions (logical OR), and `&&` means logical AND and must be kept as the last operator.

Udisks

File managers may ask for a password when trying to mount a storage device, or yield a *Not authorized* or similar error. See **Udisks#Configuration** for details.

Allow management of individual systemd units by regular users

By checking for certain values passed to the polkit policy check, you can give specific users or groups the ability to manage specific units. As an example, you might want regular users to start and stop **wpa_supplicant**:

```
/etc/polkit-1/rules.d/10-wifimanagement.rules
```

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.systemd1.manage-units") {
        if (action.lookup("unit") == "wpa_supplicant.service") {
            var verb = action.lookup("verb");
            if (verb == "start" || verb == "stop" || verb == "restart") {
                return polkit.Result.YES;
            }
        }
    }
});
```

See also

- **Polkit manual page**

Category:

- Security (Русский)

PolicyKit - создание собственных правил

Автор: А. Ракитин
Дата публикации: март 2015 г.

PolicyKit присутствует практически в любом современном дистрибутиве Linux. Он является хорошо отлаженной частью операционной системы и в то же время постоянно развивается вместе с ней. Как правило, работа PolicyKit почти незаметна для пользователя и редко требует вмешательства. Вопросы могут возникать лишь в тех случаях, когда на экране монитора появляются окна, подобные показанному ниже, и это кажется не совсем обоснованным.

Окно авторизации.

Назойливое требование ввода пароля для выполнения обыденного, хорошо понятного или часто повторяющегося действия может стать поводом для коррекции работы PolicyKit.

Немного о том, что это и как работает

PolicyKit (часто используется также сокращение PolKit) - это набор программных средств с открытым исходным кодом для гибкого управления предоставлением системных привилегий в Unix-подобных операционных системах, например, в Linux. Он позволяет непривилегированным процессам запускать привилегированные, системные. Разработчик PolicyKit - David Zeuthen. Сайт проекта http://www.freedesktop.org/wiki/Software/polkit. Лицензия - LGPL.

PolicyKit дополняет базовую модель разграничения прав доступа DAC (Discretionary Access Control), принятую в UNIX-подобных операционных системах. С небольшими упрощениями основную идею модели DAC можно изложить буквально в нескольких словах. Например, так: любой объект операционной системы является файлом и для каждого файла определены права на чтение, запись и выполнение отдельно для владельца, для членов группы и для всех остальных.

Модель DAC проста и эффективна. Но надо понимать, что она разрабатывалась во времена майнфрэймов, когда пользователь и администратор системы на самом деле были разными людьми и имели полномочия разного уровня. Очевидно, что для операционной системы, установленной на персональном компьютере требуется более гибкая модель. Для пользователя персонального компьютера такие действия как установка системного времени или даты, подключение носителей, настройка сетевого соединения, монитора или клавиатуры являются обыденными. Но по своей сути - это системные операции и они требуют прав суперпользователя. Как правило, на персональном компьютере их выполняет пользователь, который скорее всего не имеет в этот момент нужных привилегий. Но, несмотря на это, если ему все же требуется выполнить подобную операцию, то это во многих случаях не должно быть связано с необходимостью ввода административного пароля. Ввод пароля в таких ситуациях наверняка будет восприниматься как помеха, даже как признак враждебного поведения системы.

PolicyKit служит одним из способов временной и частичной передачи административных полномочий обычному, непривилегированному пользователю. Для похожих целей существуют и другие программные средства, например, sudo. Но, в отличие от sudo, PolicyKit не предоставляет администраторских полномочий на весь процесс, а следует принципу минимальных разрешений. Другими словами, он дает права суперпользователя только для выполнения конкретного действия.

Работает это следующим образом. Любой запрос на выполнение действия в системном контексте, поступивший от работающего пользовательского процесса, отслеживается с помощью PolicyKit. В соответствии с имеющимися правилами PolicyKit принимает решение о том, может ли быть выполнено это действие, и, если может, то - при выполнении каких условий. Это решение - запрет, разрешение или разрешение с условием - передается системной программе, которая затем действует соответствующим образом. Другими словами, хотя непривилегированный пользовательский процесс (Subject) и привилегированный системный процесс (Mechanism) общаются между собой напрямую, решение принимает третья сторона - PolicyKit.

Примечание. Здесь и далее по тексту в скобках приведены англоязычные термины, которые используются в документации PolicyKit и с которыми можно столкнуться при чтении статей на эту тему.

Логика работы PolicyKit.

Условием выполнения действия может быть, например, ввод оператором пароля. Для взаимодействия с оператором используется агент (Authentication agent), который при необходимости показывает оператору окно с соответствующим требованием. Агент предоставляется пользовательским окружением, например, агент PolicyKit для GNOME. Практически все современные пользовательские окружения, такие как GNOME, KDE, MATE, Xfce и другие, имеют в своем составе соответствующих агентов для взаимодействии с PolicyKit.

Все участники рассмотренного выше процесса - Mechanism, Subject, PolicyKit и Authentication agent - сообщаются между собой через системную шину сообщений D-Bus. В этом процессе может также участвовать systemd, система инициализации, которая постепенно все больше проникает во все основные составляющие операционной системы.

PolicyKit включает в себя соответствующий программный интерфейс (API), предназначенный для того, чтобы приложения могли использовать его возможности. Именно в приложениях определяются те действия (Action), которые будет отслеживать PolicyKit. На практике могут встретится программы, которые не умеют взаимодействовать с PolicyKit. Но это скорее исключение.

Для действий, которые определены в приложении, существуют соответствующие правила их выполнения (Authorization Rules). Набор таких правил для конкретного приложения называется политикой (Authorization policy).

Примерно с 2007 года PolicyKit начал использоваться во всех наиболее популярных дистрибутивах Linux и их производных - в Debian, Ubuntu, Fedora, Red Hat, OpenSUSE, Gentoo, Slackware, Archlinux и многих других. Фактически он стал стандартом в своей области.

Все изложенное ниже относится у дистрибутиву Fedora 21, но может с успехом использоваться для понимания принципов настройки PolicyKit в любых системах Linux. В данном случае использовалось пользовательское окружение MATE, но это также не принципиально.

Разработчики дистрибутивов настраивают работу PolicyKit исходя из своих представлений о безопасности. Понятно, что сервер и рабочая станция будут иметь разные настройки. При адекватном выборе дистрибутива пользователь обычно не сталкивается с какими-либо неожиданными проблемами. Но иногда все же имеет смысл внести в работу PolicyKit некоторые изменения. Это один из тех немногих случаев, когда простыми средствами можно сделать собственный компьютер чуть более отзывчивым и дружелюбным.

Явные и неявные разрешения

Явные разрешения (Explicit privileges) относятся к конкретным пользователям или группам пользователей. При этом явные разрешения могут содержать ограничения. Например, ограничением может быть требование использовать только локальную консоль.

Явные разрешения можно предоставлять или запрещать. Это похоже на всем известные списки доступа (ACL). Запрет имеет приоритет. Другими словами, если пользователю в одних политиках разрешено какое-либо действие, а в других оно запрещено, то выполнить это действие он не сможет.

Неявные разрешения (Implicit privileges) определяются для пользовательской сессии в целом. Эти разрешения, в свою очередь, могут относится к активным или к неактивным сессиям. Активная сессия - это та, в которой пользователь работает в настоящий момент.

Для принятия решения PolicyKit располагает информацией двух видов: описанием возможных **действий** и описанием **правил** для их выполнения.

Файлы действий

Список всех возможных действий содержится в файлах, которые находятся в каталоге */usr/share/polkit-1/actions*. Эти файлы записаны в формате XML, что позволяет просматривать их в текстовом редакторе или даже в браузере.

Имена файлов действий составлены из названия разработчика программного обеспечения (вендора), названия программы или группы действий и заканчиваются словом policy. Имя каждого файла вполне соответствует той группе действий, которые в нем перечислены. Средняя часть имени файла - название программы или группы действий - является в данном случае смысловой. Именно на нее надо обращать внимание, если требуется определить, в каком файле описано требуемое действие.

Для примера посмотрим на содержание файла *org.x.xf86-video-intel.backlight-helper.policy*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policyconfig PUBLIC
 "-//freedesktop//DTD PolicyKit Policy Configuration 1.0//EN"
```

```
 "http://www.freedesktop.org/standards/PolicyKit/1.0/policyconfig.dtd">
<policyconfig>
  <vendor>The X.Org project</vendor>
  <vendor_url>https://01.org/linuxgraphics/community/xf86-video-intel</vendor_url>
  <icon_name>brightness</icon_name>
  <action id="org.x.xf86-video-intel.backlight-helper">
    <description>Modify lcd panel brightness</description>
    <message>Authentication is required to modify the lcd panel brightness</message>
    <defaults>
      <allow_any>no</allow_any>
      <allow_inactive>no</allow_inactive>
      <allow_active>yes</allow_active>
    </defaults>
    <annotate key="org.freedesktop.policykit.exec.path">/usr/libexec/xf86-video-intel-
backlight-helper</annotate>
  </action>
</policyconfig>
```

Первые четыре строки - это декларация. Весь остальной текст является XML-элементом *policyconfig*. Этот элемент может быть в файле только один. В свою очередь он включат в себя следующие элементы:

- vendor - название разработчика программного обеспечения (вендора) или имя проекта (необязательный элемент).
- vendor_url - URL проекта или вендора (необязательный элемент).
- icon_name - имя значка, представляющего проект или вендора (необязательный элемент).
- action - это элемент, который содержит внутри себя другие элементы. В открывающем тэге action обязательно указывается идентификатор действия - id, который передается по шине D-Bus. Элементов action в одном файле может быть несколько. Внутри элемента action можно найти:
- description - краткое описание действия, понятное человеку. Элементов description может быть несколько, например, для разных языков.
- message - понятное оператору сообщение, которое будет появляться в окне запроса авторизации, если она потребуется. Элементов message может быть несколько, например, для разных языков.
- defaults - этот элемент определяет необходимость и тип авторизации по умолчанию для действия, указанного в элементе action. Здесь возможно использование трех необязательных элементов, определяющих неявные разрешения для клиентов, работающих в локальной консоли:
- allow_any - для любых клиентов;
- allow_inactive - для клиентов неактивных сеансов;
- allow_active - для клиентов активных сеансов;

Элементы allow_any, allow_inactive и allow_active могут содержать следующие значения:

- no - действие не разрешается;
- yes - действие разрешается;
- auth_self - требуется аутентификация от имени владельца сеанса;
- auth_admin - требуется аутентификация от имени суперпользователя, что является более серьезным ограничением, чем предыдущий вариант;
- auth_self_keep - то же, что и auth_self, но разрешение имеет силу в течение некоторого небольшого периода времени (например, пять минут);

- auth_admin_keep - то же, что и auth_admin, но разрешение имеет силу в течение некоторого небольшого периода времени (например, пять минут);
- annotate - элемент в форме "ключ-значение" используется для пояснения действия.
- vendor внутри элемента action - (в данном файле отсутствует) используется для переопределения имени проекта или вендора (необязательный элемент).
- vendor_url внутри элемента action - (в данном файле отсутствует) используется для переопределения URL проекта или вендора (необязательный элемент).
- icon_name внутри элемента action - (в данном файле отсутствует) используется для переопределения имени значка, представляющего проект или вендора (необязательный элемент).

Видно, что кроме собственно описания действия такие файлы содержат описание правил авторизации для выполнения этих действий. Причем эти правила являются правилами по умолчанию. Именно эти правила составляются разработчиками дистрибутива. Данные правила можно изменять редактированием XML-файлов .policy, но этот метод не является правильным из-за того, что при обновлении программ сделанные изменения пропадут.

Файлы и правила

Правильным методом изменения правил авторизации является использование файлов .rules, которые переопределяют правила по умолчанию из файлов действий .policy. Файлы .rules расположены в двух каталогах:

- */etc/polkit-1/rules.d* - предполагается, что здесь располагаются некоторые файлы правил, подготовленные разработчиками дистрибутива и все файлы правил, подготовленные администратором системы. При персональной настройке правил располагать соответствующие файлы надо именно в этом каталоге.
- */usr/share/polkit-1/rules.d* - данный каталог содержит файлы правил, которые написаны разработчиками приложений и дистрибутива. Размешать файлы со своими правилами здесь настоятельно не рекомендуется из-за того, что при обновлении программ сделанные изменения скорее всего пропадут.

Настраивать правила можно как правкой существующих файлов .rules, так и созданием новых - в каталоге */etc/polkit-1/rules.d*. Создание новых файлов .rules является более безопасным и надежным методом, т.к. он всегда позволяет быстро и без потерь вернуться к исходному варианту.

В Интернете можно найти материалы, описывающие настройку правил PolicyKit с помощью pkla-файлов. Это больше не актуально. **В новых версиях PolicyKit файлы правил .rules написаны на языке программирования JavaScript**.

В июне 2012 года David Zeuthen сообщил в своем блоге, что собирается переписать ту часть PolicyKit, которая работает с файлами правил. После проведения предварительных тестов он остановился на варианте с использованием языка программирования JavaScript. Такой выбор David обосновал тем, что ему хорошо знаком SpiderMonkey, интерпретатор JavaScript, а также тем, что он постоянно общается с людьми, которые имеют опыт его внедрения в GNOME Shell.

David привел достаточно веские аргументы в пользу намечавшегося кардинального изменения PolicyKit - повышение гибкости и увеличение скорости работы. Несмотря на это, по тем ответам, которые он получил, видно, что разразилась настоящая буря. Потому что минусов тоже хватало. Совершенно ясно, что среди обычных пользователей, да и среди системных администраторов Linux не так уж много тех, кто умеет программировать на JavaScript. Сам язык имел не очень хорошую репутацию в плане безопасности. К тому же считалось, что его место - Web. Кроме того, были опасения, что новый PolicyKit будет требовать установки большого количества дополнительного программного обеспечения из-за зависимостей, связанных с использованием

JavaScript. При этом, как предполагали оппоненты, возможно значительное снижение скорости его работы. Накал обсуждения был велик.

Однако, решение было принято. Новая версия PolicyKit 0.106 работала уже с новыми файлами правил. Предполагалось, что именно эта версия будет включена в дистрибутив Fedora начиная с номера 18.

Посмотреть версию установленного в системе PolicyKit можно, например,так:

```
# pkcheck --version
pkcheck version 0.112
```

На самом деле указанная в выводе версия относится к утилите pkcheck, которая входит в комплект PolicyKit. Но она же соответствует всему комплекту в целом.

Да, с некоторых пор файлы правил для PolicyKit выглядят не как традиционные файлы конфигурации, а как небольшие программы. Это несколько затрудняет написание собственных правил для тех, кто далек от программирования. Но задача облегчается тем, что во многих случаях можно использовать простой шаблон и не особенно задумываться о том как он работает. Шаблон может выглядеть, например, следующим образом:

```
// Две косые черты в начале строки обозначают комментарий.
// Здесь можно пояснить смысл правила, что является хорошей практикой.
polkit.addRule(function(action, subject) {
  if (action.id.match("действие") && subject.isInGroup("группа пользователей")) {
    return polkit.Result.правило;
  }
});
```

Конечно, это не единственный возможный шаблон. Но, похоже, один из самых простых. Немного разобравшись в конструкциях языка программирования JavaScript можно при желании составлять более сложные правила. Например, поставить выполнение действия в зависимость от таких условий как время суток или серийный номер устройства.

Шаблон очень просто использовать. Если вместо элементов, выделенных цветом, подставить нужные значения, то приведенный шаблон может быть оформлен в виде отдельного файла .rules в каталоге */etc/polkit-1/rules.d*. Ниже, в секции "Практика" рассматриваются примеры того как это можно сделать.

Для тех, кому требуется быстрый результат, а не погружение в программирование на языке JavaScript, можно дать совсем краткие пояснения того, что делает приведенный в шаблоне программный код. В данном случае используется метод addRule (), который описывает некоторую функцию. Эта функция обеспечивает проверку возможности выполнения действия и выбирает для этого нужный тип авторизации в соответствии с правилом. Это правило действительно только для определенной группы пользователей.

Осталось разобраться с тем, как правильно записать действие, группу пользователей и правило в файле .rules.

Действие - это значение id в элементе action в нужном файле действий .policy. Например, в приведенном выше файле это - org.x.xf86-video-intel.backlight-helper. При выборе нужного действия полезно внимательно прочитать его описание, приведенное в элементе description.

Группа пользователей - это одна из реально существующих в операционной системе групп. Например, это может быть та группа, которая была создана при заведении пользователя в системе.

Формулировка правил в целом совпадает с той, что используется в XML-файлах действий, которые были рассмотрены выше. Это - NO, YES, AUTH_SELF, AUTH_SELF_KEEP, AUTH_ADMIN, AUTH_ADMIN_KEEP. Они имеют тот же смысл, но в данном случае записываются в верхнем регистре.

Для того, чтобы уверенно использовать файлы .rules, надо знать, что каждый раз при запросе привилегий демон polkitd заново перечитывает файлы правил. Эти файлы читаются и обрабатываются в лексически возрастающем порядке независимо от того, в каком из двух упомянутых выше каталогов они находятся. Если файлы .rules, имеют одинаковое имя, то сначала читается файл из каталога */etc/polkit-1/rules.d*, а затем - из каталога */usr/share/polkit-1/rules.d*. В качестве иллюстрации можно привести пример того, в каком порядке будут прочитаны гипотетические файлы .rules:

```
/etc/polkit-1/rules.d/10-auth.rules
/usr/share/polkit-1/rules.d/10-auth.rules
/etc/polkit-1/rules.d/49-polkit-pkla-compat.rules
/etc/polkit-1/rules.d/50-default.rules
/usr/share/polkit-1/rules.d/lightdm.rules
```

Видно, что с помощью задания имени файла можно просто и надежно определить порядок чтения файлов .rules. На практике часто в начале имени файла записывают число. При этом файлы гарантированно будут читаться в том же порядке, в котором возрастают эти числа. Впрочем, маловероятно, что пользователю придется создавать много своих файлов с правилами. Разработчики хорошо делают свою работу и варианты по умолчанию обычно выбираются ими правильно.

Порядок чтения файлов .rules важен. Правила, которые содержатся в файлах, прочитанных раньше, переопределяют правила в файлах, прочитанных позже. Ниже этому дается объяснение.

Функции JavaScript, описывающие правила, последовательно добавляются в PolicyKit при чтении файлов .rules. Чтение происходит в том порядке, который рассматривался выше. В свою очередь, проверка правил, описанных в этих функциях, производится в том же порядке, в котором они были добавлены. Проверка производится до тех пор, пока одна из функций не возвращает значение. Другими словами, до тех пор, пока сочетание параметров действие и группа пользователей не приведет к появлению правила. В этот момент PolicyKit формирует решение о возможности выполнения действия. Если таких сочетаний в файлах правил имеется несколько, то выполняться будет первое появившееся при переборе. Это значит, что для того, чтобы добавить свое правило, его надо поместить в файл, который будет прочитан и обработан раньше других.

Продолжение статьи ...