Лабораторная работа №2. Часть 1
Углубленное исследование безопасности ядра и среды выполнения в ОС Linux

Цель работы: Изучение и настройка различных аспектов безопасности ядра Linux, включая использование AltHa, ограничение системных ресурсов и настройку параметров ядра.

1. Безопасность ядра

Ядро является основным и наиболее критическим компонентом любого дистрибутива Linux. Он обрабатывает данные программ, управляет памятью, взаимодействием между процессами, файловыми системами, сетью и всей периферией, занимается разграничением доступа, генерирует первичные сообщения аудита, может контролировать целостность программ и прочее. Следовательно, во время работы, в ядре обрабатывается или хранится множество важнейшей информации — ключи шифрования, пароли или хеши паролей (аутентификационная информация), защищаемые данные. Опасность заключается в возможности получения доступа к этой информации, обрабатывающейся ядром. Если данные неправомочно доступны, то с их помощью можно реализовать атаки любого типа.

В отличие от обычных приложений, ядро хранится в директории /boot и обычно называется **vmlinuz**.

Зайдем в учетную запись суперпользователя и посмотрим содержимое каталога:

```
# ls -l /boot
```

```
alt ~ # ls -l /boot
итого 56916
-rw-r--r-- 1 root root   243520 config-5.10.164-std-def-alt1
-rw-r--r-- 1 root root   251993 config-5.15.89-un-def-alt1
-rw-r--r-- 1 root root   155544 elf-memtest-5.31b
drwxr-xr-x 6 root root     4096 grub
-rw------- 1 root root 16022220 initrd-5.10.164-std-def-alt1.img
-rw------- 1 root root 16044374 initrd-5.15.89-un-def-alt1.img
lrwxrwxrwx 1 root root       32 initrd.img -> initrd-5.10.164-std-def-alt1.img
lrwxrwxrwx 1 root root       32 initrd-std-def.img -> initrd-5.10.164-std-def-alt1.img
lrwxrwxrwx 1 root root       30 initrd-un-def.img -> initrd-5.15.89-un-def-alt1.img
-rw-r--r-- 1 root root   153868 memtest-5.31b.bin
drwxr-xr-x 3 root root     4096 splash
-rw-r--r-- 1 root root  4859507 System.map-5.10.164-std-def-alt1
-rw-r--r-- 1 root root  5042552 System.map-5.15.89-un-def-alt1
lrwxrwxrwx 1 root root       29 vmlinuz -> vmlinuz-5.10.164-std-def-alt1
-rw-r--r-- 1 root root  7590296 vmlinuz-5.10.164-std-def-alt1
-rw-r--r-- 1 root root  7888120 vmlinuz-5.15.89-un-def-alt1
lrwxrwxrwx 1 root root       29 vmlinuz-std-def -> vmlinuz-5.10.164-std-def-alt1
lrwxrwxrwx 1 root root       27 vmlinuz-un-def -> vmlinuz-5.15.89-un-def-alt1
```

Директория /boot предназначена для хранения файлов, необходимых для загрузки ОС. Наличие нескольких файлов *vmlinuz*, каждый из которых соответствует различным версиям ядра, является стандартной практикой. Например, после обновления системы старая версия ядра не удаляется. В случае проблем с новой версией всегда загрузиться со старой, что обеспечивает дополнительную надежность и гибкость.

Чтобы узнать версию ядра, используем команду:

uname -r

```
alt ~ # uname -r
5.10.164-std-def-alt1
```

Команда **uname** отображает текущую версию ядра.

Для определения размера ядра можно использовать команду:

du -h /boot/vmlinuz-$(uname -r)

```
alt ~ # du -h /boot/vmlinuz-$(uname -r)
7,3M    /boot/vmlinuz-5.10.164-std-def-alt1
```

Команда выводит размер текущего ядра, сжатого на диске.

Стоит понимать, что внутри ядра содержится лишь основной функционал, необходимый для работы ОС. Если необходимо обеспечить дополнительный функционал, например, для работы с сетевым адаптером или видеокартой, ядро обращается к специализированным программным компонентам, известным как модули.

Модули ядра хранятся в каталоге */lib/modules*. Учитывая, что модули рассчитаны только для определенной версии ядра, то в этом каталоге создается отдельный подкаталог для каждой установленной в системе версии ядра. Зайдем в каталог текущего ядра:

ls /lib/modules/$(uname -r)

```
alt ~ # ls /lib/modules/$(uname -r)
kernel             modules.builtin          modules.dep      modules.softdep      rtl8192eu
misc               modules.builtin.alias.bin modules.dep.bin modules.symbols      rtw89
modules.alias      modules.builtin.bin      modules.devname  modules.symbols.bin  updates
modules.alias.bin  modules.builtin.modinfo  modules.order    net
```

Файл *modules.alias* содержит список всех модулей ядра с их алиасами.

Пример записи в файле *modules.alias*:

alias devname:rfkill rfkill

Когда система обнаруживает устройство с именем *devname:rfkill*, она автоматически загружает модуль *rfkill*. Модуль *rfkill* предоставляет интерфейс для включения и отключения этих устройств, а также для получения информации об их состоянии.

Посмотреть содержимое файла *modules.alias* можно с помощью следующих команд:

cat /lib/modules/$(uname -r)/modules.aliases

или

modprobe -c

Вся информация о загруженных модулях хранится в файле */proc/modules*:

cat /proc/modules

Также посмотреть загруженные модули ядра можно с помощью команды **lsmod**:

lsmod

Более подробную информацию о каждом модуле можно получить с помощью команды **modinfo**.

Чтобы узнать подробную информацию о модуле ядра *nfs*, используем команду:

modinfo nfs

В выводе можно увидеть файл модуля, его лицензию, автора и зависимости, т.е. модули, которые должны быть загружены для его нормальной работы.

Модуль ядра *nfs (Network File System)* позволяет ОС взаимодействовать с удаленными файловыми системами по сети.

Теперь загрузим модуль ядра *mtd (Memory Technology Device)* можно с помощью команд **modprobe** и **insmod**. Модуль *mtd* обеспечивает поддержку различных типов памяти, таких как флеш-память.

modprobe mtd

или

insmod /lib/modules/5.10.164-std-def-alt1/kernel/drivers/mtd/mtd.ko

Расширение **.ko** обозначает **Kernel Object** (объект ядра).

Примечание: Команде *insmod* необходимо передать полный путь до модуля *mtd*. Напомним, что его можно узнать с помощью команды *modinfo*.

modinfo mtd

```
alt ~ # modinfo mtd
filename:       /lib/modules/5.10.164-std-def-alt1/kernel/drivers/mtd/mtd.ko
description:    Core MTD registration and access routines
author:         David Woodhouse <dwmw2@infradead.org>
license:        GPL
description:    Generic support for concatenating of MTD devices
author:         Robert Kaiser <rkaiser@sysgo.de>
license:        GPL
alias:          char-major-90-*
srcversion:     D9F9F80EFB1E1DAC0FDD6AF
```

Важно! Запуск модуля ядра предпочтительно выполнять с помощью команды *modprobe*, поскольку данная команда не только находит файл модуля в файловой системе, но и загружает все его зависимости.

Удалить модуль *mtd* можно с помощью команд **modprobe** с опцией **-r** и **rmmod**.

modprobe -r mtd

или

rmmod mtd

Иногда во время загрузки системы для используемых устройств загружаются не те модули ядра либо они не поддерживают нужную функциональность либо конфликтуют с другими модулями. Чтобы решить эту проблему можно добавить модуль в чёрный список. Для этого достаточно добавить одну строчку в файл */etc/modprobe.d/blacklist.conf*.

Добавим модуль *gtp (GPRS Tunneling Protocol)* в чёрный список. Модуль *gtp* используется для передачи данных между узлами в мобильных сетях, таких как 2G, 3G и 4G.

С помощью текстового редактора Vim создадим файл */etc/modprobe.d/blacklist-gtp.conf*:

```
# vim /etc/modprobe.d/blacklist.conf
```

Добавим следующую строку:

```
blacklist gtp
```

Кроме чёрного списка существует каталог */etc/modules.load.d/*, в котором можно настроить автоматическую загрузку модулей при старте системы. Каталог содержит конфигурационные файлы с расширением *.conf*, в которых перечислены все модули, которые надо загружать при старте системы.

Добавим строку с модулем *ssb (Shared Serial Bus)* в файл */etc/modules.load.d/modules.conf*:

```
ssd
```

Собранные для определенной версии ядра модули можно просто скопировать в нужную папку, собственно, мы так и поступаем, когда собираем ядро из исходников. Однако с проприетарными драйверами и другими внешними драйверами, не поставляемыми в комплекте с ядром, дело обстоит иначе. Данные модули поддерживают несколько версий ядра и для их установки используется специальная технология — *DKMS (Dynamic Kernel Module Support)*.

При установке модуля через пакетный менеджер, *DKMS* добавляет информацию о модуле в свою базу данных. Причем модуль, установленный таким образом один раз, будет пересобираться для каждой новой версии ядра автоматически.

Правильная настройка и модификация параметров ядра повышает безопасность всей ОС. Чтобы минимизировать уязвимости и защитить систему от потенциальных атак применяют такой подход, как харденинг ядра.

В ОС “Альт Рабочая станция” данный подход реализует модуль безопасности AltHa (Alt Hardening). В настоящее время он имеет 3 варианта защиты пользовательского пространства:

1. Игнорирование битов SUID в двоичных файлах (возможны исключения).
2. Запрет на запуск определенных интерпретаторов в интерактивном режиме.
3. Отключение возможности удаления открытых файлов в выбранных каталогах.

Для включения модуля AltHa необходимо в файле загрузчика GRUB2 /etc/sysconfig/grub2 в строке GRUB_CMDLINE_LINUX_DEFAULT следует добавить опцию “altha=1”:

vim /etc/sysconfig/grub2

```
GRUB_CMDLINE_LINUX_DEFAULT=' quiet resume=/dev/disk/by-uuid/c2113ff4-c166-4e82
-8618-a797a85bba4d panic=30 splash altha=1'
```

Потом необходимо обновить загрузчик GRUB2 и перезагрузить систему:

update-grub

reboot

Теперь модуль AltHa есть в списке активных модулей безопасности:

cat /sys/kernel/security/lsm

```
alt ~ # cat /sys/kernel/security/lsm
lockdown,capability,yama,safesetid,altha,kiosk
```

Установим альтератор модуля AltHa:

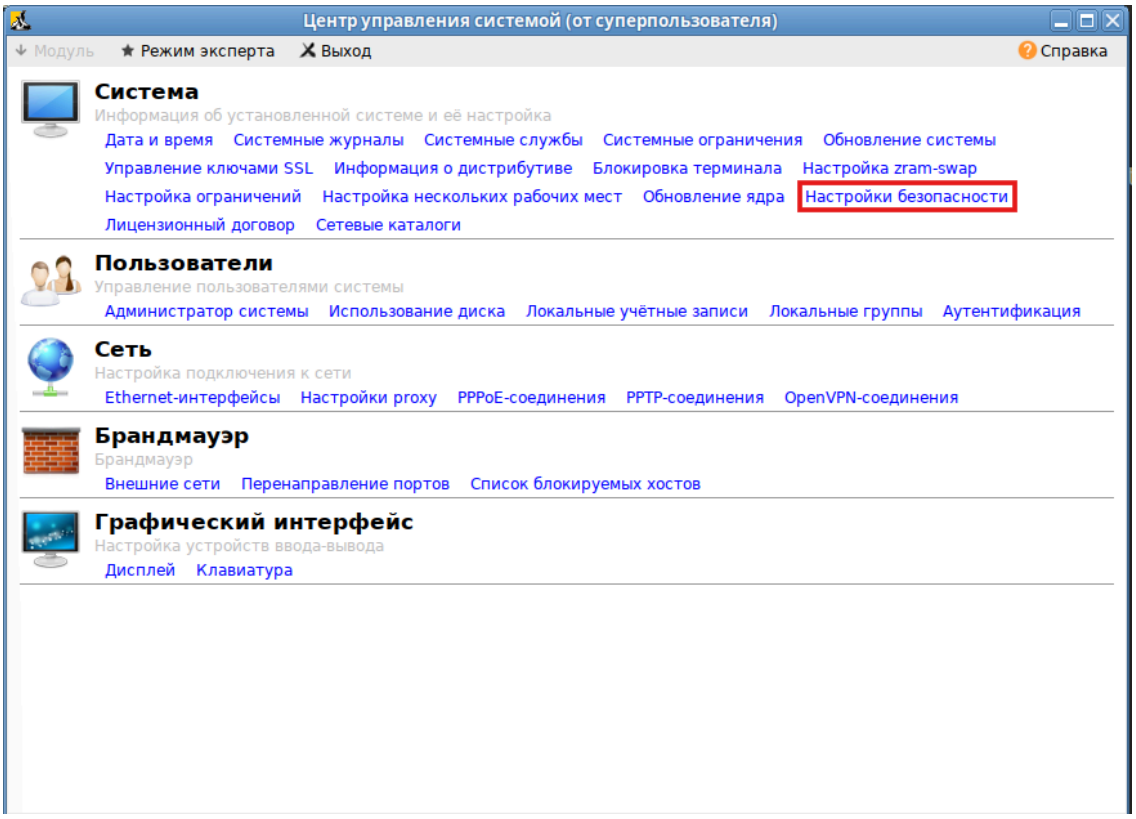
apt-get update

apt-get install alternator-secsetup

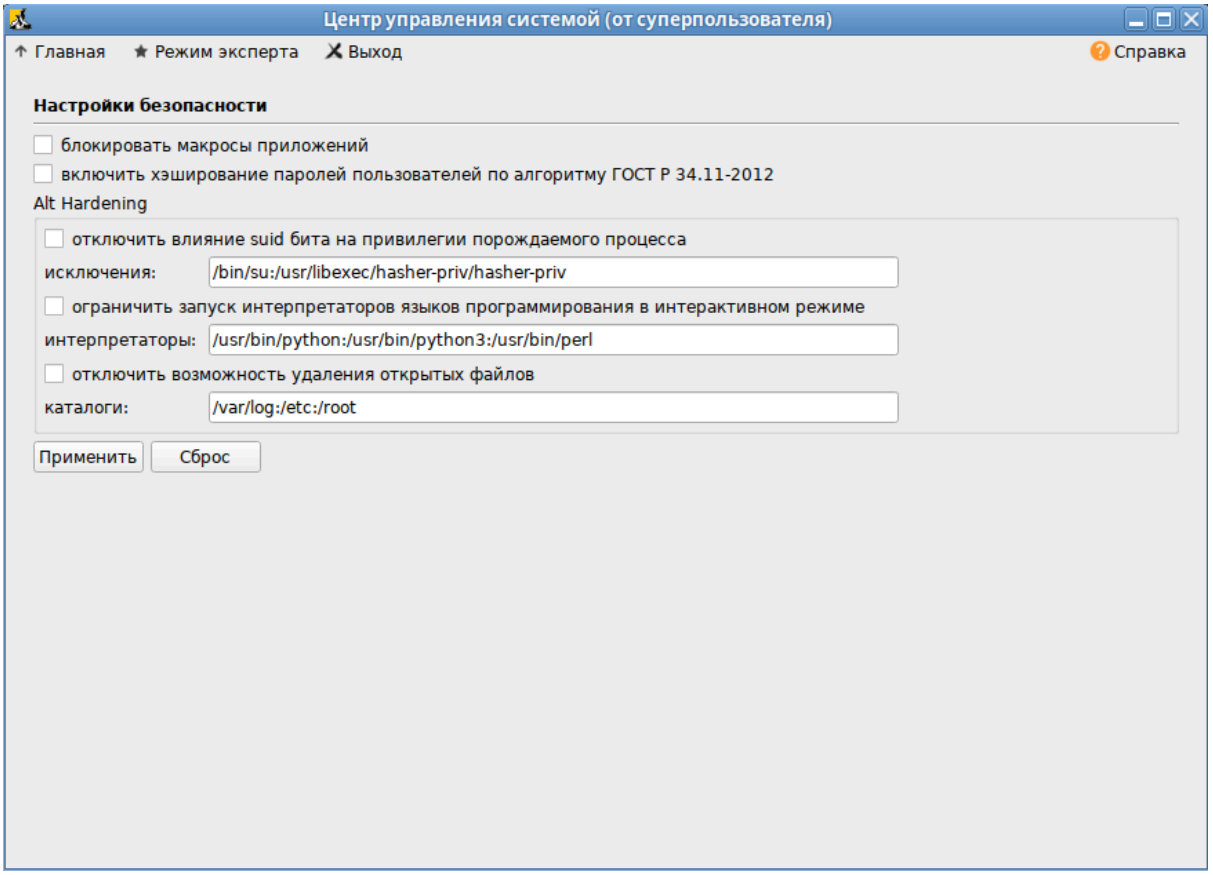
Зайдем в ЦУС (Центр управления системой):

Меню MATE → Все приложения → Администрирование → Центр управления системой.

Примечание: Альтератор потребует пароль суперпользователя.



Переходим в раздел “Настройки безопасности”:



При включенном подмодуле altha.nosuid, биты SUID во всех двоичных файлах, кроме явно перечисленных, игнорируются в масштабе всей системы.

Для включения запрета бита исполнения следует:

1. В командной строке выполнить команду:

sysctl -w kernel.altha.nosuid.enabled=1

Значения параметра: 0 — режим выключен, 1 — режим включен.

Проверка состояния режима запрета бита исполнения выполняется командой:

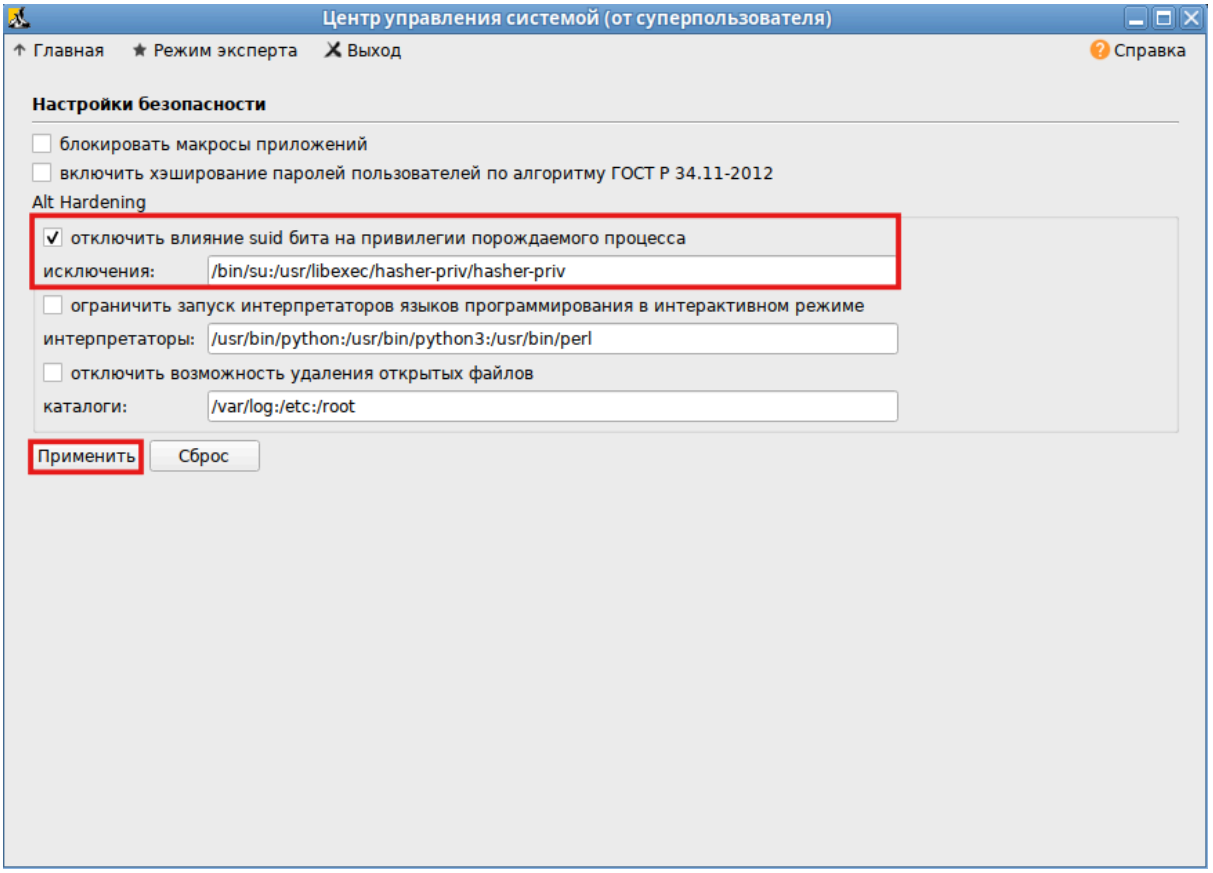
sysctl -n kernel.altha.nosuid.enabled

Добавить исключения (список включенных двоичных файлов SUID, разделенных двоеточиями) можно с помощью следующей команды:

sysctl -w kernel.altha.nosuid.exceptions=”/bin/su:/usr/libexec/hasher-priv/hasher-priv”

2. В ЦУС:

Отметить пункт “Отключить влияние suid бита на привилегии порождаемого процесса” и нажать кнопку “Применить”.



При включении блокировки интерпретаторов блокируется несанкционированное использование интерпретаторов для выполнения кода напрямую из командной строки.

Для включения блокировки интерпретаторов следует:

1. В командной строке выполнить команду:

sysctl -w kernel.altha.rstrscript.enabled=1

Значения параметра: 0 — режим выключен, 1 — режим включен.

Проверка состояния режима блокировки интерпретаторов выполняется командой:

sysctl -n kernel.altha.rstrscript.enabled

Переменная kernel.altha.rstrscript.interpreters должна содержать разделенный двоеточиями список ограниченных интерпретаторов:

sysctl -w kernel.altha.rstrscript.interpreters=”/usr/bin/python:/usr/bin/python3:/usr/bin/perl:/usr/bin/tclsh”

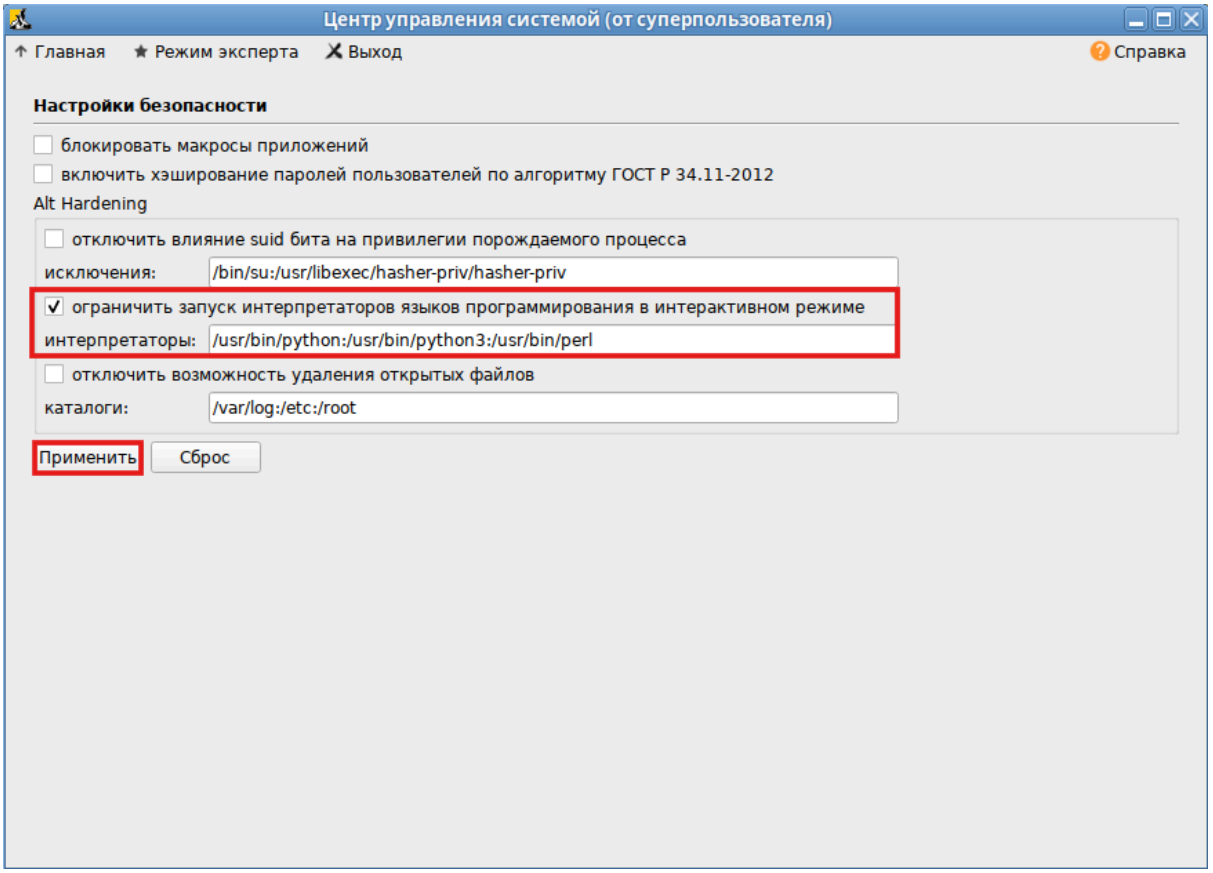
В этой конфигурации все скрипты, начинающиеся с #!/usr/bin/env python, будут заблокированы.

Вывести список заблокированных интерпретаторов можно с помощью команды:

sysctl -n kernel.altha.rstrscript.interpreters

2. В ЦУС:

Отметить пункт “Ограничить запуск интерпретаторов языков программирования в интерактивном режиме” и нажать кнопку “Применить”.



2. Ограничение системных ресурсов

Cgroups (Control Groups — контрольные группы) позволяют распределять системные ресурсы между группой задач (процессов), выполняемых в системе.

Установим все пакеты, от которых зависит работа cgroups:

```
# apt-get update
# apt-get install cgroup
```

Пакет *cgroup* предоставляет специальные утилиты командной строки, файлы конфигурации и справочные руководства для использования cgroups.

Сервис *cgconfig* используется для создания контрольных групп и управления подсистемами. Он может запускаться автоматически и восстанавливать созданные ранее *cgroups*.

Запустим сервис *cgconfig.service*:

```
# systemctl enable --now cgconfig.service
```

Проверяем статус сервиса:

```
# systemctl status cgconfig.service
```

Команда **lscgroup** позволяет вывести список текущих cgroups, включая их названия и иерархию.

```
# lscgroup
```

Системные ресурсы делятся на подсистемы (контроллеры). Каждая подсистема имеет несколько параметров, которым присваиваются различные значения.

В файле */proc/cgroups* содержится список подсистем:

cat /proc/cgroups

```
alt ~ # cat /proc/cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset  0       86      1
cpu     0       86      1
cpuacct 0       86      1
blkio   0       86      1
memory  0       86      1
devices 0       86      1
freezer 0       86      1
net_cls 0       86      1
perf_event      0       86      1
net_prio        0       86      1
hugetlb 0       86      1
pids    0       86      1
```

Значения столбцов:

subsys_name — название подсистемы cgroups;

hierarchy — количество иерархий cgroups для данной подсистемы;

num_groups — количество существующих групп cgroups, использующих данную подсистему;

enabled — указывает, включена/выключена ли данная подсистема.

Подсистема	Описание
cpuset	Ограничение и привязка наборов процессов к определенным ядрам процессора и наборам памяти.
cpu	Установка лимитов на использованием процессорного времени для групп процессов, а также управление приоритетом.
cpuacct	Статистика по использованию CPU.
blkio	Ограничение ввода/вывода блочных устройств.
memory	Управление использование ОЗУ.
devices	Контроль доступа к устройствам.
freezer	“Заморозка” и “разморозка” процессов в группе.
net_cts	Добавление сетевых меток к пакетам, отправляемым процессами в группе.
perf_event	Установка ограничений по событиям производительности для процессов.

net_prio	Установка приоритетов для сетевых соединений.
hugetlb	Управление использованием “больших страниц” (huge pager), что может улучшить производительности приложений, требующих больших объемов памяти.
pids	Лимит на максимальное количество процессов в группе.

Управлять подсистемами cgroups можно с помощью конфигурационного файла */etc/cgconfig.conf*.

cat /etc/cgconfig.conf

```
alt ~ # cat /etc/cgconfig.conf
#
#  Copyright IBM Corporation. 2007
#
#  Authors:     Balbir Singh <balbir@linux.vnet.ibm.com>
#  This program is free software; you can redistribute it and/or modify it
#  under the terms of version 2.1 of the GNU Lesser General Public License
#  as published by the Free Software Foundation.
#
#  This program is distributed in the hope that it would be useful, but
#  WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
#
# By default, mount all controllers to /sys/fs/cgroup/<controller>
#mount {
#       cpuset = /sys/fs/cgroup/cpuset;
#       cpu = /sys/fs/cgroup/cpu;
#       cpuacct = /sys/fs/cgroup/cpuacct;
#       memory = /sys/fs/cgroup/memory;
#       devices = /sys/fs/cgroup/devices;
#       freezer = /sys/fs/cgroup/freezer;
#       net_cls = /sys/fs/cgroup/net_cls;
#}

# By default, we expect systemd mounts everything on boot,
# so there is not much to do.
# See man cgconfig.conf for further details, how to create groups
# on system boot using this file.
```

В зависимости от содержимого этого файла сервис *cgconfig* может создавать иерархии, монтировать файловые системы, создавать контрольные группы и задавать параметры (ограничения ресурсов) подсистем для каждой группы.

Также управлять подсистемами можно с помощью других утилит пакета *cgconfig*.

С помощью команды **cgcreate** создадим *cgroup* с именем *lmt* с ограничением по CPU/RAM:

cgcreate -g cpu,memory:lmt

где cpu,memory — ресурсы, которые будут ограничены.

С помощью команды **cgset** установим ограничение по использованию RAM:

cgset -r memory.max=1G lmt

Также установим ограничение по использованию CPU приоритета до ~40% (1024 — 100% приоритет):

cgset -r cpu=400 lmt

Проверим создание lmt с помощью команды *lscgroup*:

lscgroup | grep lmt

Запустить новый процесс в созданной группе с помощью команды **cgexec**:

cgexec -g cpu,memory:lmt sleep 100 &

```
alt ~ # cgexec -g cpu,memory:lmt sleep 100 &
[1] 6073
```

Таким образом процесс с PID 6073 будет ограничен в использовании памяти до 1 Гб и в использовании CPU до 40%.

Проверим, запущен ли процесс внутри *lmt*. Для этого выведем список запущенных процессов в группе *lmt*:

cat /sys/fs/cgroup/cpu/lmt/cgroup.procs

```
alt ~ # cat /sys/fs/cgroup/lmt/cgroup.procs
6073
```

Сервис **cgred** перемещает задачи в cgroup согласно параметрам файла */etc/cgrules.conf*:

cat /etc/cgrules.conf

```
alt ~ # cat /etc/cgrules.conf
# /etc/cgrules.conf
#The format of this file is described in cgrules.conf(5)
#manual page.
#
# Example:
#<user>          <controllers>   <destination>
#@student        cpu,memory      usergroup/student/
#peter           cpu             test1/
#%               memory          test2/
# End of file
```

Поле <*user*> задает имя пользователя или группы.

Поле <*controllers*> задает список подсистем через запятую.

Поле <*destination*> задает имя процесса или полный путь к нему.

В записях файла также можно использовать следующие обозначения:

@ указывает на группу. Например, @wheel указывает на всех пользователей группы wheel, а не на конкретного пользователя с таким именем.

* и % задают всех пользователей.

Запустим сервис *cgred.service*:

```
# systemctl enable --now cgred.service
```

Проверяем статус сервиса:

```
# systemctl status cgred.service
```

Ограничим ресурсы веб-браузера firefox. Для этого отредактируем файл /etc/cgrules.conf:

```
# vim /etc/cgrules.conf
```

В конец файла добавим следующую запись:

```
*:firefox       cpu,memory  lmt/
```

Таким образом, процесс firefox, запущенный любым пользователем, будет добавлен в группе lmt, которая ограничит подсистемы cpu и memory.

Для проверки откроем через меню MATE веб-браузер Firefox.

Далее с помощью команды *pgrep* выведем PID процесса firefox:

```
# pgrep firefox
```

```
alt ~ # pgrep firefox
6748
6751
```

Посмотрим, какие процессы с какими PID запущены внутри *lmt*.

Для этого выведем список запущенных процессов в группе *lmt*:

```
# cat /sys/fs/cgroup/cpu/lmt/cgroup.procs
```

```
alt ~ # cat /sys/fs/cgroup/lmt/cgroup.procs
6748
6751
6823
6876
6910
6956
6957
6980
```

В списке присутствуют PID процессов веб-браузера Firefox.

3. Настройка параметров ядра

Ядро позволяет настраивать множество параметров для оптимизации работы системы. Базовые настройки определяются на этапе компиляции ядро. Тем не менее, мы также можем настраивать параметры ядра без необходимости перезагрузки или перекомпиляции.

Многие параметры ядра доступны для настройки через виртуальную файловую систему *proc*, а именно через каталог */proc/sys*.

Для изменения параметров в */proc/sys* можно использовать команду *echo* с перенаправлением в файл.

Чтобы включить пересылку IP-пакетов, можно выполнить следующую команду:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Однако внесенные изменения не сохраняются после перезагрузки системы. Чтобы сохранить настройки, можно использовать утилиту **sysctl**.

Утилита sysctl позволяет управлять параметрами ядра, представив их в виде переменных. Переменные соответствуют пути к соответствующему файлу в каталоге */proc/sys*, но вместо слеша используется точка.

Например, параметр для включения пересылки IP-пакетов будет представлен в следующем виде:

net.ipv4.ip_forward

С помощью команды *sysctl* изменим параметр *vm.swappiness*:

```
# sysctl vm.swappiness=10
```

Параметр *vm.swappiness* варьируется от 0 до 100 и определяет, насколько активно система будет использовать swap-пространство. Чем выше значение, тем больше система будет полагает на swap, даже если у нее достаточно оперативной памяти. По умолчанию установлено значение 60.

Как и в предыдущем способе изменение будет временным и не сохраняться после перезагрузки. Чтобы сделать изменение постоянным, необходимо добавить параметр в файл конфигурации */etc/sysctl.conf*.

Отредактируем файл */etc/sysctl.conf*:

```
# vim /etc/sysctl.conf
```

Добавим строку:

```
vm.swappiness=10
```

После изменения файла необходимо применить настройки:

```
# sysctl -p
```

Чтобы узнать текущее значение параметра *vm.swappiness*, можно использовать следующую команду:

```
# sysctl vm.swappiness
```

С помощью опции -a можно посмотреть все доступные параметры ядра:

```
# sysctl -a
```