SETCAP

Утилита для управления возможностями файлов в Linux.

Синтаксис

```
setcap [-q] [-n <rootuid>] [-v] {capabilities|-|-r} filename [ ... capabilitiesN fileN ]
```

Описание

Без опции **-v** (verify - проверка) **setcap** устанавливает указанные возможности для каждого файла, заданного параметром filename. Необязательный аргумент **-n <rootuid>** можно использовать для установки возможности, используемого лишь в пространстве имен пользователя, владеющего идентификатором root. Опция **-v** служит для проверки связывания указанных возможностей с файлом. При указании опций **-v** и **-n** проверяется также аргумент **-n <rootuid>**.

Возможности указываются с помощью **cap_from_text**, как описано ниже.

Функция **cap_from_text()** выделяет и инициализирует состояние возможности в рабочем хранилище. Функция устанавливает содержимое вновь созданного состояния возможности в соответствии с понятной человеку строкой символов в стиле языка C (nul-завершение), указанной buf_p и возвращает указатель на созданное состояние.
Текстовое представление возможности состоит из одного или множества разделенных пробелами «слов», каждое из которых указывает те или иные операции из набора возможностей, которые включаются или отключаются после выполнения команды (по умолчанию все отключено).

Каждое слово представляет собой список разделенных запятыми имен возможностей (или all), за которым следует список действий. Список действий включает последовательность пар «оператор-флаги». Операторами могут служить **=**, + и -, а флагами - **e**, **i** и **p**. Регистр флагов имеет значение и они указывают группу (набор), к которой относятся возможности Effective, Inheritable (наследуется) и Permitted (разрешено), соответственно.

В списке имен регистр символов не учитывается. Специальное имя **all** указывает все возможности, что эквивалентно заданию полного списка возможностей. Безымянные возможности можно указывать номером. Это позволяет библиотеке libcap поддерживать возможности, которые еще не были выделены к моменту компиляции библиотеки.

Оператор **=** задает для указанного имени сначала сброс всех 3 параметров возможности, а затем их установку в соответствии с флагами (не обязательны для этого оператора). Например, all=p сбросит установку Effective и Inheritable для батарейного питания и установит Permitted, а cap_fowner=ep установит в Effective и Permitted возможности смены владельца файла (override-file-ownership) и сбросит их в Inheritable.

При указании первым оператора **=** без списка возможностей предполагается воздействие на все возможности (**all**). Например, варианты **all=**, **=**, и **cap_chown,<every-other-capability>=** будут эквивалентны.

Операторы **+** и **-** требуют явного указания впереди списка возможностей, а после оператора - явных флагов. Оператор **+** включает все указанные в списке возможности в соответствии с флагами, а оператор **-** отключает. Например, **all+p** включает для всех возможностей Permitted, а **cap_fowner-i** отключает override-file-ownership в группе Inheritable.

Список действий может включать множество пар «оператор-флаги», действия применяются слева направо. Например, **cap_fowner+p-i** эквивалентно **cap_fowner+p cap_fowner-i**, а **cap_fowner+pe-i** эквивалентно **cap_fowner=+pe**.

Специальная строка возможностей **-** позволяет указать возможности, считываемые со стандартного устройства ввода (stdin). В таких случаях набор возможностей завершается пустой строкой.

Специальная строка **-r** служит для удаления набора возможностей файла. Отметим, что установка пустого набора возможностей **отличается** от удаления набора. Пустой набор может использоваться для предотвращения исполнения файла с привилегиями, несмотря на то, что превалирующие наборы Ambient и Inheritable позволили бы это.

Флаг **-q** служит для сокращения выводимых программой сведений.

Список возможностей

Для выполнения проверки прав доступа в традиционных системах UNIX процессы делят на две категории - привилегированные (идентификатор эффективного пользователя равен 0, как у root), и непривилегированные (ID эффективного пользователя не равен 0). Для привилегированных процессов проверки прав в ядре не выполняются, а для не привилегированных процессов выполняется полная проверка на основе возможностей процесса (обычно, эффективные UID и GID, а также и список дополнительных групп).

В ядре Linux, начиная с версии 2.2, все привилегии, обычно связываемые с суперпользователем, разделены на несколько частей, называемых возможностями (capability), которые можно разрешать и запрещать независимо. Возможности являются атрибутами потоков (нитей). Ниже приведен список возможностей, реализованных в Linux с указанием операции и поведения, разрешаемых ими. В скобках указаны версии ядра, начиная с которых каждая возможность поддерживается.

CAP_AUDIT_CONTROL (2.6.11)
: Включение и выключение аудит ядра, изменение фильтрующих правил аудита, чтение состояния аудита и фильтрующие правила.

CAP_AUDIT_READ (3.16)
: Чтение протокола аудита через многоадресный сокет netlink.

CAP_AUDIT_WRITE (2.6.11)
: Запись данных в журнал аудита ядра.

CAP_BLOCK_SUSPEND (3.5)

Возможности, которые могут приводить к блокированию приостановки системы (**epoll EPOLLWAKEUP**, /proc/sys/wake_lock).

CAP_BPF (5.8)

Привилегированные операции BPF (**bpf**, **bpf_helpers**). Возможность добавлена в Linux 5.8 для разгрузки перегруженной возможности CAP_SYS_ADMIN.

CAP_CHECKOUT_RESTORE (5.9)

- Обновление /proc/sys/kernel/ns_last_pid (**pid_namespaces**);
- использование свойства set_tid в **clone3**;
- чтение содержимого символьной ссылки /proc/[pid]/map_files для других процессов.

Возможность добавлена в Linux 5.9 для переноса функциональности checkpount/restote из CAP_SYS_ADMIN.

CAP_CHOWN

Произвольные изменения UID и GID для файлов.

CAP_DAC_OVERRIDE

Пропуск проверки доступа к файлу на чтение, запись и выполнение (DAC или discretionary access control - избирательный контроль доступа).

CAP_DAC_READ_SEARCH

- Пропуск проверки доступа к файлу на чтение и к каталогу на чтение и выполнение (просмотр);
- вызов **open_by_handle_at**;
- использование **linkat** с флагом **AT_EMPTY_PATH** для создания ссылки на файл, заданным дескриптором.

AP_FOWNER

- Пропуск проверки доступа для операций, которые обычно требуют совпадения UID файловой системы процесса и UID файла (например, **chmod**, **utime**), исключая операции, охватываемые **CAP_DAC_OVERRIDE** и **CAP_DAC_READ_SEARCH**, системы процесса и UID файла (например, **chmod utime**), исключая операции, охватываемые **CAP_DAC_OVERRIDE** и **CAP_DAC_READ_SEARCH**;
- изменение флагов inode (**ioctl_iflags**) у произвольных файлов;
- установка списков контроля доступа (ACL) для произвольных файлов;
- игнорирование закрепляющего бита при удалении файла;
- установка **O_NOATIME** для произвольных файлов в **open** и **fcntl**.

CAP_FSETID

Позволяет не сбрасывать биты режима set-user-ID и set-group-ID при изменении файла любыми дополнительными GID вызывающего процесса.

CAP_IPC_LOCK

Блокировка памяти (**mlock**, **mlockall**, **mmap**, **shmctl**).

CAP_IPC_OWNER

Отмена проверки доступа для операций с объектами System V IPC.

CAP_KILL

ioctl с операцией **KDSIGACCEPT**.

CAP_LEASE

(2.4) Установка аренды для произвольных файлов (**fcntl**).

CAP_LINUX_IMMUTABLE

Установка флагов inode **FS_APPEND_FL** и **FS_IMMUTABLE_FL** (**ioctl_iflags**).

CAP_MAC_ADMIN

(2.6.25) Изменение MAC или состояния. Реализована в Smack Linux Security Module (LSM).

CAP_MAC_OVERRIDE

(2.6.25) Замещение мандатного контроля доступа (MAC). Реализована в Smack LSM.

CAP_MKNOD

(2.4) Создание специальных файлов с помощью **mknod**.

CAP_NET_ADMIN

Различные сетевые операции:

- настройка интерфейса;
- управление IP МСЭ, трансляцией адресов и ведением учёта;
- изменение таблиц маршрутизации;
- привязка к любому адресу для организации прозрачного прокси;
- назначение типа обслуживания (TOS);
- сброс статистики драйвера;
- управление режимом захвата (promiscuous);
- включение групповой рассылки (multicasting);
- использование **setsockopt** для включения параметров сокета **SO_DEBUG**, **SO_MARK**, **SO_PRIORITY** (для приоритетов вне диапазона 0 - 6), **SO_RCVBUFFORCE** и **SO_SNDBUFFORCE**.

CAP_NET_BIND_SERVICE

Привязка сокета к привилегированным портам домена Internet (номера портов меньше 1024).

CAP_NET_BROADCAST

(не используется) Позволяет выполнять широковещание с сокета и прослушивание многоадресных рассылок.

CAP_NET_RAW

Использование сокетов RAW и PACKET и привязка к любому адресу для создания прозрачного прокси.

CAP_PERFMON (5.9)

Управление механизмами мониторига, включая:

- вызов **perf_event_open**;
- реализация операций BPF, влияющих на производительность.

Возможность добавлена в Linux 5.9 для разгрузки перегруженной возможности CAP_SYS_ADMIN.

CAP_SETGID

Произвольные действия с GID процесса и списком дополнительных GID, подмена GID при передаче возможностей сокета через доменные сокеты UNIX, запись отображения идентификатора пользователя в пользовательское пространство (**user_namespaces**).

CAP_SETFCAP

(2.6.24) Устанавливает произвольные возможности для файла.

CAP_SETPCAP

Если поддерживаются возможности для файла, позволяет добавлять любую возможность из ограничивающего набора вызывающего потока в наследуемый набор, отменять возможности из ограничивающего набора (с помощью prctl с операцией PR_CAPBSET_DROP), изменять флаги securebits.

Если возможности для файла не поддерживается (до 2.6.24), позволяет предоставлять и отменять любую возможность из списка разрешённых вызывающего или любого другого процесса (свойство недоступно с ядрами, поддерживающими возможности для файлов, поскольку CAP_SETPCAP в таких ядрах имеет другую семантику).

CAP_SETUID

Произвольные действия с UID процесса (**setuid**, **setreuid**, **setresuid**, **setfsuid**), подмена UID при передаче возможностей сокета через доменные сокеты UNIX, запись отображения идентификатора пользователя в пользовательское пространство (**user_namespaces**).

CAP_SYS_ADMIN

- Управление системой, включая **quotactl**, **mount**, **umount**, **swapon**, **swapoff**, **sethostname**, **setdomainname**;
- привилегированные операции syslog (начиная с Linux 2.6.37, для этих операций нужно использовать CAP_SYSLOG);
- команды VM86_REQUEST_IRQ vm86;
- функциональность checkpoint/restore, обеспечиваемая CAP_CHECKPOIBT_RESTORE (предпочтительна);
- функциональность BPF, обеспечиваемая CAP_BPF (предпочтительна);
- функциональность мониторинга производительности, обеспечиваемая CAP_PERFMON (предпочтительна);
- операции IPC_SET и IPC_RMID над произвольными объектами System V IPC;
- перезапись ограничения ресурса RLIMIT_NPROC;
- операции над расширенными атрибутами trusted и security (**xattr**);
- использование **lookup_dcookie**;
- использование **ioprio_set** для назначения классов планирования ввода-вывода IOPRIO_CLASS_RT и (до Linux 2.6.25) IOPRIO_CLASS_IDLE;
- подмена PID при передаче возможостей сокета через доменные сокеты UNIX;
- превышение системного ограничения (/proc/sys/fs/file-max) на количество открытых файлов в системных вызовах, открывающих файлы (например, **accept**, **execve**, **open**, **pipe**);
- использование флагов CLONE_*, создающих новые пространства имён с помощью **clone** и **unshare** (начиная с Linux 3.8 для создания пользовательских пространств никаких возможностей не требуется);
- вызовы **perf_event_open**;
- доступ к информации о привилегированном событии perf;
- вызовы **setns** (требуется CAP_SYS_ADMIN в целевом пространстве имён);
- вызовы **fanotify_init**;
- привилегированные операции KEYCTL_CHOWN и KEYCTL_SETPERM в **keyctl**;
- вызовы **ptrace** PTRACE_SECCOMP_GET_FILTER для получения дампа фильтров seccomp трассируемого;
- операция MADV_HWPOISON в **madvise**;
- использование TIOCSTI в **ioctl** для вставки символов во входную очередь терминала, отличного от управляющего терминала вызывающего;
- устаревший системный вызов **nfsservctl**;
- устаревший системный вызов **bdflush**;
- привилегированные операции **ioctl** над блочными устройствами;
- привилегированные операции **ioctl** над файловой системой;
- привилегированные операции **ioctl** над устройством /dev/random (**random**);
- установка фильтров **seccomp** без начальной установки атрибута потока no_new_privs;
- изменение правил разрешения и запрета для групп управления устройствами;
- операция **ptrace** PTRACE_SECCOMP_GET_FILTER для получения дампа фильтров seccomp трассируемого;
- операция **ptrace** PTRACE_SETOPTIONS для приостановки защиты seccomp трассируемого (т. е., флаг PTRACE_O_SUSPEND_SECCOMP)
- административные операции над многими драйверами устройств;
- изменение значений приоритета autogroup путем записи в /proc/[pid]/autogroup.

CAP_SYS_BOOT

Использование **reboot** и **kexec_load**.

CAP_SYS_CHROOT

Использование **chroot** и смена примонтированных пространств имен с помощью **setns**.

CAP_SYS_MODULE

Загрузка и выгрузка модулей ядра (**init_module** и **delete_module**), в ядрах до версии 2.6.25 - отзыв возможностей из системного набора Bounded.

CAP_SYS_NICE

- Снижение приоритета процесса (**nice**, **setpriority**) и изменение приоритета произвольных процессов;
- задание политики планирования в реальном масштабе времени для вызывающего процесса, а также политики планирования и приоритетов для произвольных процессов (**sched_setscheduler**, **sched_setparam**, **shed_setattr**);
- привязка к ЦП для произвольных процессов (**sched_setaffinity**);
- назначение класса планирования ввода-вывода и приоритета для произвольных процессов (**ioprio_set**);
- применение **migrate_pages** к произвольным процессам для их переноса на произвольные узлы;
- применение **move_pages** к произвольным процессам;
- использование флага **MPOL_MF_MOVE_ALL** в **mbind** и **move_pages**.

CAP_SYS_PACCT

Использование **acct**.

CAP_SYS_PTRACE

- Трассировка любого процесса с помощью **ptrace**;
- применение **get_robust_list** к произвольным процессам;
- перенос данных в память произвольного процесса (или из нее) с помощью **process_vm_readv** и **process_vm_writev**;

- изучение процессов с помощью **kcmp**.

CAP_SYS_RAWIO

- Операции ввода-вывода из портов (**iopl** и **ioperm**);
- доступ к /proc/kcore;
- использование операции **FIBMAP** в **ioctl**(2);
- создание устройств для доступа к специальным регистрам x86 (MSR, **msr**);
- обновление /proc/sys/vm/mmap_min_addr;
- отображение памяти по адресам меньше значения, заданного в /proc/sys/vm/mmap_min_addr;
- отображение файлов в /proc/bus/pci;
- доступ к /dev/mem и /dev/kmem;
- выполнение различных команд устройств SCSI;
- определённые операции с устройствами **hpsa** и **cciss**;
- некоторые специальные операции с другими устройствами.

CAP_SYS_RESOURCE

- Использование зарезервированного пространства файловых систем ext2;
- вызовы **ioctl**, управляющие журналом ext3;
- превышение ограничений дисковой квоты;
- увеличение ограничений по ресурсам (**setrlimit**);
- перезапись ограничений ресурсов **RLIMIT_NPROC**;
- превышение максимального числа консолей при выделении консоли;
- превышение максимального числа раскладок;
- использование более 64hz прерывания из часов реального времени;
- установка значения msg_qbytes очереди сообщений System V больше /proc/sys/kernel/msgmnb (**msgop** и **msgctl**);
- разрешение RLIMIT_NOFILE ограничивать число файловых дескрипторов, передаваемых в обход, при передаче другому процессу через доменный сокет UNIX (**unix**);
- переопределение /proc/sys/fs/pipe-size-max при назначении вместимости канала с помощью команды **F_SETPIPE_SZ** в **fcntl**;
- использование **F_SETPIPE_SZ** для увеличения вместимости канала сверх /proc/sys/fs/pipe-max-size;
- переопределение /proc/sys/fs/mqueue/queues_max, /proc/sys/fs/mqueue/msg_max, /proc/sys/fs/mqueue/msg-size_max при создании очередей сообщений POSIX (**mq_overview**);
- операция **prctl PR_SET_MM**();
- установка в /proc/[pid]/oom_score_adj значения меньше последнего, заданного процессом с помощью CAP_SYS_RESOURCE.

CAP_SYS_TIME

Настройка системных часов (**settimeofday**, **stime**, **adjtimex**) и часов реального времени (аппаратных).

CAP_SYS_TTY_CONFIG

Использование **vhangup**(2) и привилегированные операции **ioctl** с виртуальными терминалами.

CAP_SYSLOG (2.6.37)

Привилегированные операции **syslog**, просмотр адресов ядра в /proc и других интерфейсах, когда значение /proc/sys/kernel/kptr_restrict = 1 (kptr_restrict в **proc**).

CAP_WAKE_ALARM (3.0)

Вызов чего-либо при пробуждении системы (установка таймеров **CLOCK_REALTIME_ALARM** и **CLOCK_BOOTTIME_ALARM**).

Группы (наборы) возможностей потока

Каждый поток (нить) имеет несколько групп (наборов) возможностей из числа перечисленных выше. Группа может быть пустой.

Permitted – разрешенные возможности

Ограничивающее надмножество набора Effective с возможностями, которые поток может предполагать. Этот набор также ограничивает список возможностей, которые могут быть добавлены в наследуемый набор потоком без **CAP_SETPCAP** в своём наборе Effective.

Если поток отменяет возможность в своем наборе Permitted, он не сможет вернуть ее (без вызова **execve** из программы с set-user-ID-root или программы, чьи возможности для файла позволяют это).

Inheritable – наследуемые возможности

Этот набор возможностей, сохраняемых при вызове **execve**. Наследуемые возможности остаются таковыми при выполнении любой программы и добавляются в набор Permitted, если у выполняющейся программы установлены соответствующие биты в наследуемом наборе для файлов.

Поскольку наследуемые возможности обычно не сохраняются после **execve**, если выполнение происходит не от суперпользователя, приложениям, которым нужно выполнять вспомогательные программы с расширенными возможностями, нужно применять внешние возможности (Ambient), описанные ниже.

Bounding

(на уровне потока, начиная с Linux 2.6.25) Механизм, который может использоваться для ограничения возможностей, предоставляемых вызовом **execve**.

Начиная с Linux 2.6.25, этот набор задается на уровне потока, а в более ранних версиях это атрибут системы для всех потоков.

Effective

Набор возможностей, используемый ядром при выполнении проверок прав для потока.

Ambient - внешние (охватывающие) возможности

(начиная с Linux 4.3) Набор возможностей, сохраняемый после **execve** для непривилегированных программ. Для набора внешних возможностей соблюдается правило, в соответствии с которым возможность не может быть внешней, если она не входит в оба набора Permitted и Inheritable.

Набор внешних возможностей можно изменять с помощью **prctl**. Этот набор автоматически сужается при сужении соответствующего набора Permitted и Inheritable.

Выполнение программы, меняющей UID или GUI из-за установленного бита set-user-ID или set-group-ID, а также програмы, которая может устанавливать возможности для файла, сбрасывает набор Ambient. Внешние возможности добавляются к набору Permitted и назначаются набору Effective при вызове **execve**. Если внешние

возможности расширяют набор Permitted или Effective при вызове **execve**, это не вызывает режим защищенного исполнения (secure-execution), описанный в **ld.so**.

Потомки (ветви) потока, созданные с помощью **fork**, наследуют копии наборов родительских возможностей.

Поток может менять свои возможности с помощью **capset**.

Начиная с версии 2.6.24 файл /proc/sys/kernel/cap_last_cap показывает максимальное численное значение возможностей, поддерживаемых работающим ядром. Это позволяет определить старший бит, который можно установить в наборе возможностей.

Возможности для файлов

Начиная с ядра 2.6.24 поддерживается связывание наборов возможностей с исполняемым файлом с помощью утилиты **setcap**. Наборы возможностей хранятся в расширенном атрибуте (**setxattr**, **xattr**) с именем security.capability. Для записи в этот атрибут требуется возможность CAP_SETFCAP. Наборы возможностей файла вместе с наборами возможностей потока определяют реальные возможности потока после вызова **execve**.

Для файла поддерживатся три набора возможностей.

Permitted (ранее forced)
Возможности, автоматически предоставляемые потоку, независимо отнаследуемых им возможностей.

Inheritable (ранее allowed)
Пересечение (AND) этого набора с набром наследуемых потоком возможностей определяет для потока набор Permitted после вызова **execve**.

Effective
По сути, это не набор возможностей, а просто бит, при установке которого вызов **execve** переносит все вновь разрешенные для потока возможности в новый набор Effective. При сброшенном бите ни одна из вновь разрешенных вызовом **execve** возможностей потока не присутствует в новом наборе Effective.
Установка бита предполагает, что любая из возможностей Effective или Inheritable для файла, которая заставляет поток обретать соответствующую возможность Permitted в результате вызова **execve** (Правила преобразования), также будет включена в набор Effective. Поэтому при включении возможности для файла (**setcap**, **cap_set_file**, **cap_set_fd**) установка бита Effective для любой возможности должна сопровождаться установкой этого флага для всех других возможностей, где установлен флаг Permitted или Inheritable.

Правила преобразования

При выполнении **execve** ядро рассчитывает новые возможности процесса, как показано ниже.

```
P'(ambient)     = (файл привилегированный) ? 0 : P(ambient)
P'(permitted)   = (P(inheritable) & F(inheritable)) | (F(permitted) & P(bounding)) | P'(ambient)
P'(effective)   = F(effective) ? P'(permitted) : P'(ambient)
P'(inheritable) = P(inheritable)    [т. е. не меняется]
P'(bounding)    = P(bounding)       [т. е. не меняется]
```

где P() указывает значение возможности потока до вызова **execve**, P'() - возможности после вызова, F() - набор возможностей для файла.

Отметим некоторые детали, относящиеся к правилам преобразования возможностей:

- набор Ambient появился лишь в Linux 4.3 и при преобразовании набора внешних возможностей в процессе **execve** привилегированным считается файл, имеющий возможности или установленный бит set-user-ID или set-group-ID bit set;
- до Linux 2.6.25 набор Bounding был системным атрибутом для всех потоков и это значение использовалось для расчетов при вызове **execve** как P(bounding).

Отметим, что при описанном выше преобразовании возможности файла можно игнорировать (считать пустыми) по тем же причинам, которые служат для игнорирования битов set-user-ID и set-group-ID. Возможности файлов игнорируются при загрузке ядра с опцией no_file_caps option.

Текст подготовлен на основании материалов man из Mageia 8.

Николай Малых

nmalykh@protocols.ru