USB Guard

Доступ к приложению только при наличии USB-ключа (VID/PID + optional serial)

Курсовой проект по ОС • Linux kernel module + launcher

Родионов Денис

Задача

Что нужно было сделать и какие ограничения важны

Требования

- Определять наличие конкретного USB-устройства (VID/PID, при необходимости serial)
- Давать понятный статус в user space (0/1)
- Запускать приложение только при наличии ключа
- Не портить систему: без dd, без чтения/записи сырых секторов

Как решено

- Модуль ядра: слушает события USB (ADD/REMOVE)
- Матч по VID/PID + serial (проверка serial только на ADD)
- Интерфейс: /proc/usb_guard возвращает "0" или "1"
- Launcher: читает /proc/usb_guard и решает запускать или нет

Архитектура

Два компонента: модуль ядра + пользовательский launcher

Модуль ядра: что происходит внутри

Почему это работает стабильно (и почему мы не читаем диск)

Алгоритм

- При init: регистрируем USB notifier (usb_register_notify)
- На USB_DEVICE_ADD: проверяем VID/PID и, если задано, serial
- Если совпало: запоминаем конкретное устройство (usb_get_dev)
- На USB_DEVICE_REMOVE: сравниваем указатель устройства и сбрасываем состояние
- Статус хранится как 0/1 (atomic) и отдаётся через /proc

Почему не читаем serial на REMOVE

При извлечении устройство уже “уходит”, и чтение строковых дескрипторов может не сработать.

Поэтому:
1) проверяем serial только на ADD
2) на REMOVE сбрасываем по “тому же устройству” (tracked usb_device*)

Безопасность

- Нет операций с /dev/sdX
- Нет dd / сырого чтения секторов
- Не зависит от ФС и разметки флешки
- Меньше риск повредить систему

Интерфейс /proc/usb_guard

Как launcher узнаёт статус

Идея

Мы создаём виртуальный файл в procfs.
Когда программа делает read(), обработчик
в модуле формирует строку "0\n" или "1\n"
и ядро возвращает её в user space.

Это простой read-only интерфейс для статуса.

Мини-фрагмент кода

```
static ssize_t proc_read(...){
  int v = present ? 1 : 0;
  int len = scnprintf(msg, sizeof(msg), "%d\n", v);
  return simple_read_from_buffer(ubuf, count, ppos, msg, len);
}

static const struct proc_ops proc_fops = {
  .proc_read = proc_read,
};
```

Пользовательская часть

Launcher: разрешить или запретить запуск

Вариант 1: скрипт

```
STATUS=$(cat /proc/usb_guard 2>/dev/null)
if [ "$STATUS" = "1" ]; then
  exec ./my_app
else
  echo "Нет USB-ключа: доступ запрещён"
  exit 1
fi
```

Вариант 2: C launcher (usb_launch.c)

Читает /proc/usb_guard → если 1, делает execl(...) на “реальный” бинарник приложения.

Плюс: можно использовать setgid-группу и закрытую папку /opt/usbkey, чтобы не запускали напрямую.

Hardening (идея)

- Кладём реальный бинарник: /opt/usbkey/app.real (chmod 750, root:usbkey)
- Лаунчер ставим в /usr/local/bin с setgid (chmod 2755, root:usbkey)
- Обычный пользователь не может запустить app.real напрямую
- Важно: root всё равно может обойти (это демонстрация контроля доступа)

Сборка и запуск (чеклист)

Команды для сборки, загрузки и проверки

```
# 1) зависимости
sudo apt update
sudo apt install -y build-essential linux-headers-$(uname -r)

# 2) сборка модуля
make

# 3) параметры USB-ключа
lsusb
udevadm info -q property -n /dev/sdX | grep SERIAL

# 4) загрузка модуля
sudo insmod usb_guard.ko vendor=0xabcd product=0x1234 serial="..."

# 5) проверка
cat /proc/usb_guard
dmesg | tail -n 50

# 6) запуск приложения через launcher
./run_with_usb_key.sh
# или
./usb_mines

# 7) выгрузка
sudo rmmod usb_guard
```

Демонстрация

План действий (по шагам)

Сценарий

- Показать VID/PID в lsusb (и serial через udevadm при необходимости)
- Загрузить модуль: insmod usb_guard.ko vendor=... product=... serial=...
- Проверить: cat /proc/usb_guard → 1 (ключ вставлен)
- Запустить приложение через launcher (usb_mines или run_with_usb_key.sh)
- Вытащить флешку: lsusb уже не показывает, cat /proc/usb_guard → 0
- Повторить запуск launcher → отказ в доступе
- Вставить флешку обратно → cat /proc/usb_guard → 1, запуск снова разрешён
- Опционально: показать hardening (app.real недоступен напрямую)

Итоги и ограничения

Что получилось и что важно честно сказать

Плюсы

- Безопасно для системы: не трогаем диски и ФС
- Простой интерфейс: /proc/usb_guard → 0/1
- Работает с любым приложением через launcher
- Надёжное снятие статуса при REMOVE (tracked usb_device*)

Ограничения

- Это контроль доступа уровня ОС, а не криптозащита
- Пользователь с root-правами сможет обойти
- VID/PID могут совпадать у одинаковых устройств → лучше добавлять serial
- Для “запрета прямого запуска” нужен hardening (права/группы)

Вопросы?

usb_guard: модуль ядра + /proc + launcher