sysctl в НАЙС.ОС

NICE.OS — базовая безопасность «из коробки» 🔒

Ниже — официальный профиль системных параметров (sysctl), который включён в NICE.OS по умолчанию. Мы разберём каждую настройку простым языком, объясним почему так, где это помогает, а где может потребоваться переопределение. Цель профиля — сделать систему безопасной «из коробки», не ломая совместимость с типичными сценариями.

✅ **Коротко:** включена строгая рандомизация памяти, скрытие адресов ядра, запрет опасных функций и жёсткая гигиена сетевого стека (anti-spoofing, без redirect/source-route, SYN-защита). Маршрутизация отключена по умолчанию.
Сборка: ООО «НАЙС СОФТ ГРУПП» • niceos@ncsgp.ru

Файл профиля по умолчанию

Параметры применяются при загрузке системы. Ниже приведён актуальный базовый набор, который поставляется с NICE.OS «из коробки» (с комментариями для понимания смысла).

```
# NICE.OS — базовые sysctl по умолчанию (с акцентом на безопасность)

# 🔐 Память и ядро
kernel.randomize_va_space = 2        # строгая ASLR
kernel.kptr_restrict = 2             # скрывать адреса ядра
kernel.dmesg_restrict = 1            # dmesg — только root
kernel.sysrq = 0                     # отключить SysRq (магические клавиши)
fs.suid_dumpable = 0                 # нет core dump для setuid/setgid

#  ptrace и защита процессов
kernel.yama.ptrace_scope = 1         # ptrace разрешён только на свои процессы

#  Ограничение PID (анти-fork-bomb)
kernel.pid_max = 65536               # верхняя граница PID по умолчанию
```

```
#  Защита от атак через /proc и IPC
fs.protected_symlinks = 1             # безопасные symlink
fs.protected_hardlinks = 1            # безопасные hardlink
fs.protected_regular = 1              # защита регулярных файлов
fs.protected_fifos = 1                # защита именованных каналов

# 🌐 Сеть: антиспуфинг и гигиена
net.ipv4.tcp_challenge_ack_limit = 1073741823  # DoS-защита TCP
net.ipv4.ip_forward = 0               # маршрутизация IPv4 — выкл по умолчанию
net.ipv6.conf.all.forwarding = 0      # маршрутизация IPv6 — выкл по умолчанию

# Anti-spoofing / RPF
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# Запрет ICMP Redirect (не посылать и не принимать)
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0

# Запрет source routing (IPv4/IPv6)
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

# SYN-защита и очереди
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_tw_reuse = 0             # не переиспользовать TIME_WAIT
```

i **Где лежит:** параметры могут быть объединены в один или несколько файлов внутри `/etc/sysctl.d/` и применяются командой `sysctl --system`.

Карта настроек: что включено и зачем 🗺️

Ниже по порядку разбираем каждую группу — без «лево/право» блоков, последовательно и подробно.

1) Память и ядро 🔒

Строгая ASLR — `kernel.randomize_va_space = 2`

Включает полную рандомизацию адресного пространства процессов. Итог: эксплойтам сложнее предсказать, где лежат библиотеки, стек, куча, ядро — атаки становятся нестабильными или невозможными.

Скрытие адресов ядра — `kernel.kptr_restrict = 2`

Не позволяет обычным пользователям и сервисам читать точные адреса символов ядра. Это убирает «дорожную карту» для эскалации привилегий. Даже root увидит их только в особых отладочных режимах.

Ограничение доступа к dmesg — `kernel.dmesg_restrict = 1`

Журнал ядра закрыт для не-root пользователей. Потому что там часто бывают технические детали, помогающие атакующему: стек вызовов, адреса, имена символов, ошибки драйверов.

Отключение SysRq — `kernel.sysrq = 0`

«Магические» комбинации SysRq умеют останавливать задачи, сбрасывать файлы, выключать систему. На сервере в продакшене — это лишний риск. Всегда можно включить временно для диагностики.

Безопасные аварийные дампы — `fs.suid_dumpable = 0`

Если setuid/setgid-программа падает, её дамп может содержать секреты. Запрещаем такие дампы — меньше шансов утечки ключей/паролей/токенов.

---

2) ptrace и защита процессов 🛡

Запрет «шпионить» за чужими процессами — `kernel.yama.ptrace_scope = 1`

Пользователь может отлаживать только собственные процессы (с тем же UID). Это снижает риск кражи секретов через `ptrace` и делает сложнее атаки класса «инъекция в соседний процесс».

Верхняя граница PID — `kernel.pid_max = 65536`

Разумный лимит на количество процессов защищает от fork-bomb и других инцидентов, когда платформа внезапно порождает слишком много процессов (например, из-за бага). Если у вас специально высокая плотность процессов —

значение можно поднять, но это должно быть осознанное решение.

Защита symlink/hardlink/regular/fifo — `fs.protected_*`

Эти флаги закрывают целый класс атак в общедоступных каталогах (вроде `/tmp`), когда злоумышленник подменяет путь на другой файл (symlink/hardlink) или внедряет «ловушку» с именованным каналом. Включённые защиты делают такие техники бесполезными.

---

3) Сетевой стек: гигиена и устойчивость 🌐

TCP Challenge ACK — `net.ipv4.tcp_challenge_ack_limit = 1073741823`

Максимально высокий лимит делает некоторые классы DoS-атак практически неприменимыми — ядро охотнее отвечает «challenge ACK» и не даёт сбивать состояние TCP сессий.

Маршрутизация по умолчанию — выключена — `net.ipv4.ip_forward = 0`, `net.ipv6.conf.all.forwarding = 0`

Сервер по умолчанию не является роутером. Это правильная модель безопасности: меньше открытых поверхностей, меньше случайных маршрутов. Если сервер должен форвардить трафик (NAT, Kubernetes-узел), включайте осознанно и только там, где это требуется.

Reverse Path Filtering (anti-spoofing) — `rp_filter = 1`

«Строгий» режим RPF отбрасывает пакеты, пришедшие не с того интерфейса, где система ожидала их увидеть по таблице маршрутизации. Это срезает целый пласт IP-спуфинга. Для сложных overlay/многих маршрутных доменов может потребоваться ослабить до `2` (loose).

Запрет ICMP Redirect и source routing

`send_redirects = 0`, `accept_redirects = 0`, `accept_source_route = 0` (и для IPv4, и для IPv6) — классическая гигиена безопасности. Никто не сможет навязать узлу «обходные» маршруты, а сам узел не будет рассылать подсказки.

Защита от SYN flood

- `net.ipv4.tcp_syncookies = 1` — включаем SYN cookies, чтобы очередь полуоткрытых соединений не захлёбывалась от мусора.
- `net.ipv4.tcp_max_syn_backlog = 4096` — увеличиваем бэклог, чтобы переживать пики входящих соединений.
- `net.ipv4.tcp_tw_reuse = 0` — не переиспользуем соединения в TIME_WAIT. Это немного консервативнее, но надёжнее.

⚠️ **Важно:** значения по умолчанию выбирались с приоритетом безопасности и предсказуемости. В сценариях с экстремальным количеством коротких TCP-сессий возможен тюнинг (см. ниже).

Практика: как жить с этим профилем в реальных сценариях

Сценарий A — обычный сервер/ВМ (web, БД, CI) 🖥️

Ничего менять не нужно. Базовый профиль обеспечивает хорошую безопасность без побочных эффектов. Если у вас много коротких TCP-соединений и наблюдаются редкие «провалы» под пиками — проверьте лимиты на сокеты и очередь в приложении, балансировку, а затем при необходимости увеличьте `tcp_max_syn_backlog`.

Сценарий B — контейнер-хост / Kubernetes-узел ☁️

В кластере маршрутизация и работы на бриджах/overlay-сетях — норма, поэтому используйте профиль *NICE.OS CLOUD* (с поддержкой форвардинга, br_netfilter, расширенных лимитов и пр.). Базовый профиль по умолчанию намеренно не включает эти опции — чтобы не открывать лишнего там, где это не требуется.

Сценарий C — узел-шлюз/NAT/роутер 🌍

Включите форвардинг и соответствующие правила firewall, ослабьте `rp_filter` до `2`, если есть асимметрия маршрутов или сложные VRF/overlay. Не забывайте журналировать и мониторить.

💡 **Принцип:** всё, что расширяет поверхность атаки (форвардинг, «loose»-проверки, ускорители TCP) — включаем только там, где это действительно приносит пользу. Без фанатизма.

Проверка: быстро убедиться, что всё применилось 🔍

Несколько команд для «самотеста» после загрузки:

```
# Память/ядро:
sysctl kernel.randomize_va_space
sysctl kernel.kptr_restrict
sysctl kernel.dmesg_restrict
sysctl kernel.sysrq
sysctl fs.suid_dumpable

# Процессы:
sysctl kernel.yama.ptrace_scope
sysctl kernel.pid_max

# Сеть (гигиена):
sysctl net.ipv4.tcp_challenge_ack_limit
sysctl net.ipv4.ip_forward
sysctl net.ipv6.conf.all.forwarding
sysctl net.ipv4.conf.all.rp_filter
sysctl net.ipv4.conf.all.accept_redirects
sysctl net.ipv4.tcp_syncookies
sysctl net.ipv4.tcp_max_syn_backlog
sysctl net.ipv4.tcp_tw_reuse
```

Где искать конфликты: если какое-то приложение меняет sysctl на лету, фиксируйте это в инфраструктурном коде (Ansible, облачный init, systemd-таски), чтобы на следующей перезагрузке всё оставалось предсказуемым.

Когда имеет смысл отступить от дефолтов (осознанно)

1) Много коротких TCP-сессий, пиковый трафик 🌊

Возможные шаги:

- Увеличить `net.ipv4.tcp_max_syn_backlog` (например, 8192–16384).
- Проверить *listen-backlog* на стороне приложения/балансировщика.
- При необходимости — тюнинг *somaxconn* и сетевых очередей (уже относится к облачному профилю).

2) Kubernetes/контейнерные нагрузки ☸

Переходите на профиль **NICE.OS CLOUD** (см. отдельную статью):

- Включён форвардинг IPv4/IPv6, `br_netfilter`, расширенные лимиты conntrack, PID/FD/inotify.
- Полезные настройки для ingress/NodePort, overlay/eBPF, мониторинга.

3) Узел-роутер/NAT, сложная маршрутизация

- Включить `ip_forward` и соответствующую фильтрацию.
- Ослабить `rp_filter` до `2` при асимметрии маршрутов.
- Вести мониторинг и аудит правил.

FAQ — частые вопросы 💬

Почему маршрутизация выключена по умолчанию? 🛡

Потому что это самый безопасный и предсказуемый режим для серверов и ВМ: узел не перенаправляет трафик и не становится неосознанно «частью сети». Если вам нужен роутинг — вы знаете, зачем и где его включить, и включите точечно.

Что даёт запрещённый `tcp_tw_reuse`?

Чуть меньшую производительность в сценариях с гигантским количеством коротких сессий — но более безопасную и предсказуемую работу TCP-стека. В большинстве серверных сценариев это оптимальный компромисс. Если нужен другой баланс — тюнингуйте осознанно.

Нельзя ли открыть доступ к `dmesg` обычным пользователям?

Технически можно, но не рекомендуется. Журнал ядра содержит чувствительную информацию, полезную для атак и отладки. Открывайте доступ временно и только на тестовых средах, либо давайте права узкой группе инженеров.

Повторяется строка `kernel.yama.ptrace_scope` — это ошибка?

Нет: повторное указание одного и того же значения вреда не приносит. Итоговое значение будет тем, что указано последним. В нашем дефолте оно равно `1`.

Запомнить: базовый профиль NICE.OS — это «безопасная отправная точка». Он не мешает обычным серверным сценариям и не превращает систему в роутер. Всё, что расширяет поверхность — включается сознательно.

Итог 🏁

Базовые настройки NICE.OS закрывают типичные векторы атак: рандомизируют память, скрывают адреса ядра, защищают `/proc`, ограничивают возможности `ptrace`, выключают опасные функции и приводят сетевой стек в порядок: антиспуфинг, запрет redirect/source-route, защита от SYN flood.

За счёт выключенной маршрутизации по умолчанию система остаётся компактной и безопасной. Когда потребуется роль роутера или контейнер-хоста — используйте профиль **NICE.OS CLOUD** и включайте дополнительные возможности точечно.

Такой подход остаётся понятным, управляемым и «инфраструктура-как-код»: дефолты — безопасны, расширения — осознанны. 🖤