Hardening-флаги в НАЙС.ОС

В НАЙС.ОС по умолчанию включены ключевые hardening-флаги, которые значительно повышают устойчивость системы к эксплуатации уязвимостей. Эти флаги активируют защитные механизмы на уровне бинарных файлов: PIE (перемещаемые исполняемые файлы), RELRO (защита таблиц GOT), SSP (Stack Smashing Protection), ASLR (рандомизация адресного пространства), FORTIFY_SOURCE (дополнительные проверки функций libc). Вместе они делают практически невозможным эксплуатацию типовых багов вроде переполнения буфера, возврата в libc или ROP-атак без сложных обходов. Эти защиты активированы как для системных сервисов, так и для пользовательских приложений, включая sudo, sshd, rpm, gpg. Пользователи получают безопасную по умолчанию систему, а разработчики — возможность собирать устойчивые к атакам приложения без лишних усилий.

Hardening-флаги в НАЙС.ОС

Техническая документация: политика сборки, механизм защиты, проверка соответствия, применение в проектах
Версия документа: 1.0
Статус: действующий
Аннотация
Документ устанавливает технические требования к применению компиляторных и линковочных механизмов усиления безопасности (hardening) в НАЙС.ОС и в пользовательских проектах. Рассматриваются: модель угроз, перечень механизмов, типовые флаги сборки, методы верификации (checksec, hardening-check, readelf), интеграция в RPM-сборку и CI/CD, ограничения и порядок исключений.

1. Область применения

Требования настоящего документа распространяются на:

- системные пакеты и сервисы НАЙС.ОС, собираемые в RPM-окружении;
- прикладные бинарные файлы и библиотеки, распространяемые в составе решений на базе НАЙС.ОС;
- пользовательские проекты на C/C++ (а также смешанные проекты), собираемые на НАЙС.ОС.

Документ не заменяет регуляторные требования (сертификация, применение СКЗИ, контроль целостности и т.п.). Hardening рассматривается как инженерная мера снижения эксплуатационности уязвимостей класса memory corruption.

2. Термины, определения и сокращения

Термин	Определение
Hardening	Совокупность мер компиляции/линковки/настроек исполнения, повышающих устойчивость ПО к эксплуатации уязвимостей.
PIE	Position Independent Executable. Исполняемый файл, допускающий загрузку по произвольному адресу (условие эффективного ASLR).
ASLR	Address Space Layout Randomization. Рандомизация размещения сегментов процесса (stack/heap/libs/text и др.).
RELRO	Relocation Read-Only. Перевод таблиц релокаций/GOT в режим «только чтение» после разрешения символов.
SSP	Stack Smashing Protector. Защита стека с применением «канарейки» (stack canary) и аварийным завершением при повреждении.
FORTIFY_SOURCE	Усиленные проверки некоторых функций libc (копирование/строки/память) при наличии оптимизаций и информации о размерах буферов.
NX	Non-eXecutable memory. Запрет исполнения кода из страниц данных (включая стек), при поддержке аппаратного NX-бита и настроек линкера.

Термин	Определение
CET	Control-flow Enforcement Technology. Аппаратные механизмы защиты управления потоком (в зависимости от платформы/ядра/компилятора).

3. Модель угроз и цель hardening

Hardening направлен на снижение успешности эксплуатации уязвимостей, связанных с повреждением памяти и управлением потоком исполнения, включая:

- переполнение буфера (stack/heap), выход за границы массива (OOB);
- use-after-free (UAF), двойное освобождение, повреждение метаданных аллокатора;
- подмена адресов переходов/вызовов функций (return-to-libc, ROP/JOP);
- перезапись таблиц динамической линковки (GOT/PLT) и релокаций;
- выполнение внедрённого кода из данных (shellcode) при ошибках в проверках границ.

Примечание
Hardening не устраняет причину уязвимости и не заменяет безопасную разработку, статический/динамический анализ, обновления и контроль цепочки поставки. Цель hardening — увеличить стоимость атаки и перевести часть классов эксплуатации в отказ по безопасности (fail closed) либо в неэксплуатируемое состояние.

4. Механизмы hardening: состав и назначение

В НАЙС.ОС hardening рассматривается на трёх уровнях: компиляция, линковка, исполнение. Механизмы должны применяться согласованно.

Механизм	Назначение / эффект	Типовая реализация (флаги/настройки)
PIE	Обеспечивает возможность загрузки бинарного кода по произвольному адресу, повышая эффективность ASLR.	`-fPIE` (compile) + `-pie` (link)

Механизм	Назначение / эффект	Типовая реализация (флаги/настройки)
ASLR	Рандомизирует адресное пространство процесса; снижает предсказуемость адресов gadget/функций/структур.	Параметры ядра; эффективность повышается при PIE
RELRO	Усложняет подмену адресов динамических символов (GOT/relocations). В режиме «Full» фиксирует разрешение символов на старте процесса.	`-Wl,-z,relro` + `-Wl,-z,now`
SSP	Обнаруживает повреждение стека и завершает процесс до перехода по перезаписанному адресу возврата.	`-fstack-protector-strong` (или эквивалентный профиль)
FORTIFY	Добавляет проверки для части API libc при наличии оптимизаций и информации о размерах буферов.	`-D_FORTIFY_SOURCE=2` (и выше) + оптимизация (`-O`)
NX	Запрещает исполнение кода из стека и части областей данных при поддержке аппаратного NX и корректных атрибутов ELF.	Линковка: `-Wl,-z,noexecstack`; контроль ELF сегмента GNU_STACK
No RPATH	Снижает риски подмены библиотек через небезопасные пути загрузки.	Контроль: отсутствие RPATH/RUNPATH в ELF
Format security	Повышает вероятность выявления ошибок форматных строк на этапе компиляции.	`-Wformat -Werror=format-security` (политика проекта)

Внимание
Hardening является «сквозной» настройкой. Частичное применение (например, SSP без PIE/RELRO) снижает суммарный эффект и усложняет контроль соответствия. Рекомендуется применять профильным набором флагов, централизованно, через макросы сборки и CI-проверки.

5. Политика НАЙС.ОС: применение по умолчанию

В НАЙС.ОС hardening-флаги рассматриваются как базовая характеристика поставки. Системные пакеты должны собираться с включёнными защитными механизмами по

умолчанию. Применение обеспечивается:

- глобальными RPM-макросами для компилятора и линковщика;
- единым набором %optflags и глобальных LDFLAGS;
- контролем соответствия в QA/CI для критических пакетов и образов.

Администратор может проверить эффективные флаги окружения сборки:

Команды проверки макросов RPM

```
rpm --eval '%{optflags}'
rpm --eval '%{__global_cflags}'
rpm --eval '%{__global_cxxflags}'
rpm --eval '%{__global_ldflags}'
```

Примечание
Для пакетов с нестандартной системой сборки допускается явная прокладка флагов через переменные окружения (CFLAGS/CXXFLAGS/LDFLAGS) при условии сохранения профиля hardening и прохождения проверки.

6. Проверка включённых механизмов

Проверка должна выполняться на уровне готовых ELF-объектов (исполняемые файлы и разделяемые библиотеки), а также на уровне параметров системы (для ASLR и смежных механизмов).

6.1 Автоматическая проверка (checksec)

Утилита checksec предоставляет агрегированное состояние защит ELF (RELRO, canary, NX, PIE, Fortify и др.).

Пример: проверка бинарного файла

```
checksec --file /usr/bin/sudo
```

Типовой набор атрибутов, который должен присутствовать у критических системных бинарников:

- RELRO: Full
- Canary: Yes

- NX: Enabled
- PIE: Enabled
- FORTIFY: Enabled (для динамических сборок при поддержке libc)

Примечание
Реальный формат вывода зависит от конкретной реализации checksec. В ряде реализаций также отображаются поля RPATH/RUNPATH, наличие CFI и количество «fortified» вызовов.

6.2 Автоматическая проверка (hardening-check)

Утилита hardening-check применяется для быстрого контроля соответствия набору требований hardening (характерна для практик дистрибутивной сборки и аудит-процедур).

Пример: проверка бинарного файла

```
hardening-check /usr/bin/ssh
hardening-check /usr/sbin/sshd
```

6.3 Ручная верификация ELF (readelf)

Проверка PIE (тип ELF):

PIE: ELF type

```
readelf -h /usr/bin/ssh | egrep 'Type:|Entry point'
```

Для PIE характерен тип ET_DYN (в отличие от ET_EXEC у не-PIE исполняемых файлов).

Проверка RELRO (наличие сегмента GNU_RELRO):

RELRO: GNU_RELRO

```
readelf -l /usr/bin/ssh | grep -F 'GNU_RELRO' || true
```

Проверка «Full RELRO» (принудительное раннее связывание символов):

Full RELRO: BIND_NOW

```
readelf -d /usr/bin/ssh | egrep 'BIND_NOW|FLAGS' || true
```

Проверка NX стека (GNU_STACK должен быть без флага E):

NX: GNU_STACK

```
readelf -W -l /usr/bin/ssh | grep -F 'GNU_STACK' || true
```

6.4 Проверка ASLR (параметры ядра)

Для систем общего назначения обычно используется режим полной рандомизации адресного пространства. Проверка выполняется через sysctl-интерфейс ядра:

ASLR: randomize_va_space

```
cat /proc/sys/kernel/randomize_va_space
```

Примечание
Настройка ASLR зависит от профиля системы (включая контейнерные окружения) и политик эксплуатации. Значение параметра следует фиксировать в конфигурации профиля и контролировать средствами аудита.

7. Подключение hardening в собственных проектах

Рекомендуется централизованно задавать флаги через переменные окружения и/или настройки системы сборки. Ниже приведены минимально достаточные примеры. Конкретный профиль должен соответствовать политике сборки НАЙС.ОС.

7.1 Makefile

Makefile: CFLAGS/LDFLAGS

```
CFLAGS  += -O2 -pipe -fstack-protector-strong -D_FORTIFY_SOURCE=2 -fPIE
LDFLAGS += -Wl,-z,relro -Wl,-z,now -Wl,-z,noexecstack -pie
```

```
all: app
app: main.o
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
```

7.2 CMake

CMakeLists.txt: добавление флагов

```
cmake_minimum_required(VERSION 3.16)
project(app C)

add_executable(app main.c)

target_compile_options(app PRIVATE
  -O2 -pipe
  -fstack-protector-strong
  -D_FORTIFY_SOURCE=2
  -fPIE
)

target_link_options(app PRIVATE
  -Wl,-z,relro
  -Wl,-z,now
  -Wl,-z,noexecstack
  -pie
)
```

7.3 Meson

meson.build: project arguments

```
project('app', 'c', default_options : ['buildtype=release'])

add_project_arguments(
  '-O2', '-pipe',
  '-fstack-protector-strong',
  '-D_FORTIFY_SOURCE=2',
  '-fPIE',
  language: 'c'
)

add_project_link_arguments(
  '-Wl,-z,relro',
  '-Wl,-z,now',
  '-Wl,-z,noexecstack',
```

```
  '-pie',
  language: 'c'
)

executable('app', 'main.c')
```

7.4 Autotools

configure: переменные окружения

```
export CFLAGS="-O2 -pipe -fstack-protector-strong -D_FORTIFY_SOURCE=2 -fPIE"
export LDFLAGS="-Wl,-z,relro -Wl,-z,now -Wl,-z,noexecstack -pie"

./configure
make -j"$(nproc)"
make install
```

Внимание
Для разделяемых библиотек следует использовать -fPIC (а не -fPIE). Для статических сборок применимость отдельных механизмов (FORTIFY, RELRO, checksec-атрибуты) должна проверяться отдельно.

8. Ограничения и типовые конфликтные случаи

8.1 Производительность и старт процесса

- RELRO + -z now может увеличивать время старта процессов за счёт раннего разрешения символов.
- PIE добавляет косвенные издержки на адресацию; эффект зависит от архитектуры и профиля нагрузки.
- FORTIFY добавляет проверки, которые проявляются на горячих путях только при фактическом использовании защищаемых API.

8.2 Совместимость и особенности сборок

- Некоторые низкоуровневые компоненты (загрузчики, рантаймы, JIT, особые статические сборки) могут требовать специализированных профилей.
- Плагины, загружаемые сторонним рантаймом, требуют согласования флагов (PIE/PIC, visibility, LTO) на уровне всего графа зависимостей.

- FORTIFY требует оптимизаций компилятора (как минимум -O1) и поддержки соответствующей libc.

8.3 Отладка

PIE/ASLR усложняют анализ адресов при отладке. Для воспроизводимости допускается временное отключение рандомизации в отладочных сессиях (в рамках регламента разработки), без изменения production-профиля.

GDB: управление рандомизацией

```
# пример для отладочной сессии
gdb -q ./app
(gdb) set disable-randomization on
(gdb) run
```

9. Контроль соответствия: CI/CD и аудит

Контроль соответствия должен быть автоматизирован и выполняться на артефактах сборки. Минимальный контроль включает:

- проверку критических бинарников на PIE/RELRO/NX/SSP/FORTIFY;
- контроль отсутствия RPATH/RUNPATH;
- проверку, что профиль сборки не деградировал при изменениях спеков/toolchain.

9.1 Пример: GitHub Actions

.github/workflows/hardening.yml

```
name: hardening-check

on:
  push:
  pull_request:

jobs:
  hardening:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```

```
    - name: Build
      run: |
        make -j"$(nproc)"

    - name: Check hardening (example)
      run: |
        checksec --file ./app || true
        hardening-check ./app || true
```

9.2 Пример: пакетная проверка каталога

Сканирование каталога с бинарниками

```
set -euo pipefail

BIN_DIR="${1:-./bin}"

find "$BIN_DIR" -type f -maxdepth 1 -print0 | while IFS= read -r -d '' f; do
 if file "$f" | grep -q 'ELF'; then
   echo "==> $f"
   checksec --file "$f" || true
   hardening-check "$f" || true
 fi
done
```

Примечание
В production-контуре рекомендуется сохранять отчёты проверки (артефакты CI) и обеспечивать трассируемость: версия исходников параметры сборки результаты hardening-проверок опубликованный пакет/образ.

10. Порядок исключений и документирование отклонений

Отклонение от профиля hardening допускается только при наличии технического обоснования и при соблюдении процедуры:

1. фиксируется причина (несовместимость, требование real-time, ограничение рантайма);
2. описывается область действия (конкретный бинарник/библиотека/модуль);
3. определяются компенсирующие меры (изоляция, sandbox, ограничение привилегий, конфигурация сервиса);

4. в CI добавляется отдельный контроль, подтверждающий, что исключение не расширилось на другие артефакты.

Внимание
Неформализованные исключения (например, «так проще собрать») не допускаются. Любое отключение SSP/PIE/RELRO/NX для компонентов, обрабатывающих недоверенные данные, должно считаться повышением риска и требовать отдельного согласования.

11. План развития

Политика hardening рассматривается как непрерывно развиваемая. При наличии поддержки со стороны toolchain и ядра могут расширяться следующие направления:

- аппаратные механизмы защиты управления потоком (включая CET — при наличии аппаратной и программной поддержки);
- усиление профилей компиляции для части C/C++ библиотек (включая дополнительные проверки стандартной библиотеки);
- расширение набора автоматических проверок (включая анализ RPATH/RUNPATH, запрет небезопасных флагов линковки);
- интеграция результатов hardening-сканирования в отчётность поставки (SBOM/аттестационные артефакты), при необходимости.

12. Приложения

12.1 Контрольный список (минимальный профиль)

Параметр	Критерий соответствия
PIE	Исполняемые файлы: ET_DYN (за исключением документированных исключений)
RELRO	Наличие GNU_RELRO; для критических компонентов — «Full» (с BIND_NOW)
NX	GNU_STACK без флага исполнения; отсутствие исполняемого стека
SSP	Наличие символов/паттернов stack canary (проверка checksec/hardening-check)

Параметр	Критерий соответствия
FORTIFY	Включено при поддержке libc и наличии оптимизаций; контроль выборочно по критическим пакетам
RPATH/RUNPATH	Отсутствует, если не требуется по архитектуре решения

12.2 Приложение: шаблон отчёта проверки

Шаблон отчёта (пример)

```
# Объект проверки:
#   Пакет/компонент:
#   Версия:
#   Сборка (git hash / release):
#   Дата:
#
# Команды:
#   checksec --file <path>
#   hardening-check <path>
#   readelf -h/-l/-d <path>
#
# Результаты:
#   PIE:
#   RELRO:
#   NX:
#   SSP:
#   FORTIFY:
#   RPATH/RUNPATH:
#
# Заключение:
#   Соответствует / Не соответствует
#   Отклонения (если есть) + обоснование
```

12.3 Ссылки (для сопровождения документа)

- ld(1) — описание линковочных опций семейства -z (RELRO, NOW, NOEXECSTACK и др.).
- checksec — инструмент проверки защит ELF (RELRO, NX, PIE, Fortify и др.).
- hardening-check — утилита быстрого контроля hardening-профиля.

- _FORTIFY_SOURCE — механизм усиленных проверок libc.

Заключение

Hardening-флаги являются обязательным инженерным уровнем защиты для системного и прикладного ПО в НАЙС.ОС. Корректная реализация включает: единый профиль сборки, верификацию артефактов, автоматический контроль в CI, формализованный порядок исключений. Такой подход снижает эксплуатационность типовых уязвимостей класса memory corruption без изменения функциональности программного кода.

Источники

- `ld(1)` man page: https://man7.org/linux/man-pages/man1/ld.1.html
- Пример вывода checksec (свойства RELRO/NX/PIE/Fortify): https://lib.rs/crates/checksec
- Описание практик Linux hardening (PIE/ASLR и др.): https://book.hashbang.sh/docs/security/linux_hardening/
- Документация hardening-check (Debian): (поиск/описание пакета) https://packages.debian.org/
- Документация glibc по FORTIFY_SOURCE: (справочные материалы glibc) https://sourceware.org/glibc/