?????? ????????????? ?????

ldd ./program

```
$ ldd /bin/ls
        linux-vdso.so.1 (0x00007fff2b9fe000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f3cdd5b3000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3cdd1b0000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f3cdd7e0000)
```

Показывает список динамических библиотек, которые будут загружены при запуске программы.
ldd фактически выполняет часть программы через загрузчик, нельзя запускать ldd на подозрительных бинарниках.

readelf -d
Читает динамическую таблицу ELF-файла и показывает список библиотек, от которых он зависит.

```
$ readelf -d /usr/bin/mate-calc | grep NEEDED
 0x0000000000000001 (NEEDED)             Совм. исп. библиотека: [libgtk-3.so.0]
 0x0000000000000001 (NEEDED)             Совм. исп. библиотека: [libgdk-3.so.0]
 0x0000000000000001 (NEEDED)             Совм. исп. библиотека: [libpango-1.0.so.0]
 0x0000000000000001 (NEEDED)             Совм. исп. библиотека: [libatk-1.0.so.0]
```

objdump -p

Аналогично readelf, но может показать больше контекста.

```
objdump -p /usr/bin/mate-calc | grep
NEEDED
  NEEDED               libgtk-3.so.0
  NEEDED               libgdk-3.so.0
  NEEDED               libpango-1.0.so.0
  NEEDED               libatk-1.0.so.0
```

В рантайме — strace

Если вы хотите узнать, какие библиотеки реально подгружаются при запуске (в том числе через dlopen() во время работы):

```
strace -e open ./program 2>&1 | grep '\.so'
```

Если программа статически слинкована (т.е. все зависимости включены внутрь ELF), то ни ldd, ни readelf не покажут зависимостей.

Проверить это можно так:

```
file ./program
```

Пример:

```
# динамическая линковка
file /usr/bin/mate-
calc

/usr/bin/mate-calc: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=58d991a9b15b10d59d235c79b2132fde101cf1fc, for GNU/Linux 3.2.0, stripped

# статическая линковка
... statically linked ...
```

Пример анализа зависимостей для /usr/bin/grep.

1. Проверим, какой это тип ELF-файла

```
file /usr/bin/grep
/usr/bin/grep: ELF 64-bit LSB executable, x86-64, dynamically linked, interpreter \
/lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, stripped
```

Значит, это динамически слинкованный ELF, и у него есть внешние зависимости.

2. Смотрим список зависимостей через ldd

```
ldd /usr/bin/grep
        linux-vdso.so.1 (0x00007fff43dfe000)
        libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f3f5a8b0000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3f5a4a0000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f3f5aaf0000)
```

Интерпретация:

- libpcre2-8.so.0 — библиотека регулярных выражений PCRE2.

- libc.so.6 — стандартная библиотека C (glibc).
- ld-linux-x86-64.so.2 — динамический загрузчик ELF.

3. Альтернатива: readelf -d

```
readelf -d /usr/bin/grep | grep NEEDED
 0x0000000000000001 (NEEDED)             Shared library: [libpcre2-8.so.0]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

Это подтверждает те же зависимости.

4. Проверим, какие библиотеки реально подгружаются при запуске (в том числе через dlopen())

```
strace -e openat /usr/bin/grep "test" /etc/passwd 2>&1 | grep '\.so'
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpcre2-8.so.0", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

5. Проверим динамический загрузчик (интерпретатор ELF)

```
readelf -l /usr/bin/grep | grep 'interpreter'
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

Это та программа, которая первым делом загружает бинарь и его библиотеки.

Итог:

Тип проверки	Инструмент	Что показывает
Тип ELF	`file`	Статически или динамически слинкован
Динамические зависимости	`ldd`, `readelf -d`, `objdump -p`	Список библиотек
Реальные загрузки во время работы	`strace`	Библиотеки, подгружаемые во время исполнения
ELF-интерпретатор	`readelf -l`	Путь к динамическому загрузчику

Можно проверить, какие именно функции приложение импортирует из библиотек.

Теперь углубимся на уровень **символов (функций)**, которые исполняемый файл импортирует из библиотек — например, какие функции `grep` реально использует из `libc`, `libpcre2` и других.

1. Проверим импортированные символы (динамически загружаемые функции)

Для этого можно использовать:

```
readelf -Ws /usr/bin/grep | grep ' UND '
```

или короче:

```
readelf -Ws /usr/bin/grep | grep UND | less
```

- `-W` — показывает полные строки.
- `-s` — выводит таблицу символов.
- `UND` означает *undefined symbol* — то есть функция **определена в внешней библиотеке** (т.е. импортируется).

Пример вывода (сокращённо)

```
Num:    Value  Size Type    Bind   Vis      Ndx Name
  45: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND strcmp@GLIBC_2.2.5
  72: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND malloc@GLIBC_2.2.5
  89: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND regexec@LIBPCRE2_8
 114: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND free@GLIBC_2.2.5
 132: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fprintf@GLIBC_2.2.5
```

Расшифровка:

- `strcmp`, `malloc`, `free`, `fprintf` → из `libc`
- `regexec` → из `libpcre2-8.so.0`

---

2. Посчитать, сколько внешних символов программа импортирует

Можно примерно оценить так:

```
readelf -Ws /usr/bin/grep | grep ' UND ' | wc -l
```

Например, может вывести:

```
125
```

значит, `grep` использует около 125 внешних функций из библиотек.

---

3. Уточнить, из какой именно библиотеки берётся символ

Команда `ldd` показывает пути библиотек, а `readelf` — имена функций, но если ты хочешь **увидеть, какой именно `.so` экспортирует нужную функцию**, можно воспользоваться `objdump` или `nm`.

Пример: узнать, где определена функция `regexec`

```
for lib in /lib/x86_64-linux-gnu/*.so*; do
    nm -D $lib 2>/dev/null | grep ' regexec$' && echo "→ $lib"
done
```

Результат:

```
000000000001f230 T regexec
→ /lib/x86_64-linux-gnu/libpcre2-8.so.0
```

---

4. Альтернатива: `objdump -T`

Если хочешь посмотреть **все экспортируемые символы библиотеки**:

```
objdump -T /lib/x86_64-linux-gnu/libc.so.6 | grep printf
```

Это покажет строку вроде:

```
0000000000054350 g    DF .text  00000000000001a5  GLIBC_2.2.5 printf
```

Резюме

Задача	Команда	Что показывает
Список импортированных функций	`readelf -Ws program	grep UND`
Кол-во импортов	`readelf -Ws program	grep UND
Где определён символ	`nm -D /lib/...	grep <имя>`
Все экспортируемые символы из .so	`objdump -T libname.so`	Список функций, предоставляемых библиотекой

Revision #2
Created 2 November 2025 17:23:37 by Admin
Updated 2 November 2025 18:00:19 by Admin