Позиционно-независимый код
ASLR
dlopen/dlsym

Программа на низком уровне

.o - это бинарный файл, который делится на секции:

```
заголовок     - содержит сводную информацию
.text         - двоичный код модуля
.data         - инициализированные глоб. перем.
.bss          - неинициализированные глоб. перем.
.rodata       - константы

.debug***           - отдадочная информация
.note .comments     - служебная информация
```

Программа на низком уровне

foo.c

```
int var = 10;
void foo1 () {
}
void foo2 () {
  foo1();
}
```

bar.c

```
extern int var;
extern void foo2();
extern void bar () {
  int loc = var;
  foo2();
}
```

foo.o

```
0x0010: int var = 10
0x0014: void foo1 ()
. . . .
0x00A0: void foo2 () {
        call foo1 # 0x0014
0x00A2: ret }
```

bar.o

```
0x0000: void bar () {
        alloca loc
0x0002: loc = foo.var
0x0004: call foo.foo2
0x0006: ret }
```

Программа на низком уровне

foo.c

```
int var = 10;
void foo1 () {
}
void foo2 () {
  foo1();
}
```

bar.c

```
extern int var;
extern void foo2();
extern void bar () {
  int loc = var;
  foo2();
}
```

foo_bar.elf

```
0x0010: int var = 10
0x0014: void foo1 ()
. . . .
0x00A0: void foo2 () {
        call foo1 # 0x0014
0x00A2: ret }
0x00A4: void bar () {
        alloca loc
0x00A6: loc = var # 0x0010
0x00A8: call foo2 # 0x00A0
0x00AA: ret }
```

Программа на низком уровне

foo_bar.elf

```
0x0010: int var = 10
0x0014: void foo1 ()
. . . .
0x00A0: void foo2 () {
        call foo1 # 0x0014
0x00A2: ret }
0x00A4: void bar () {
        alloca loc
0x00A6: loc = var # 0x0010
0x00A8: call foo2 # 0x00A0
0x00AA: ret }
```

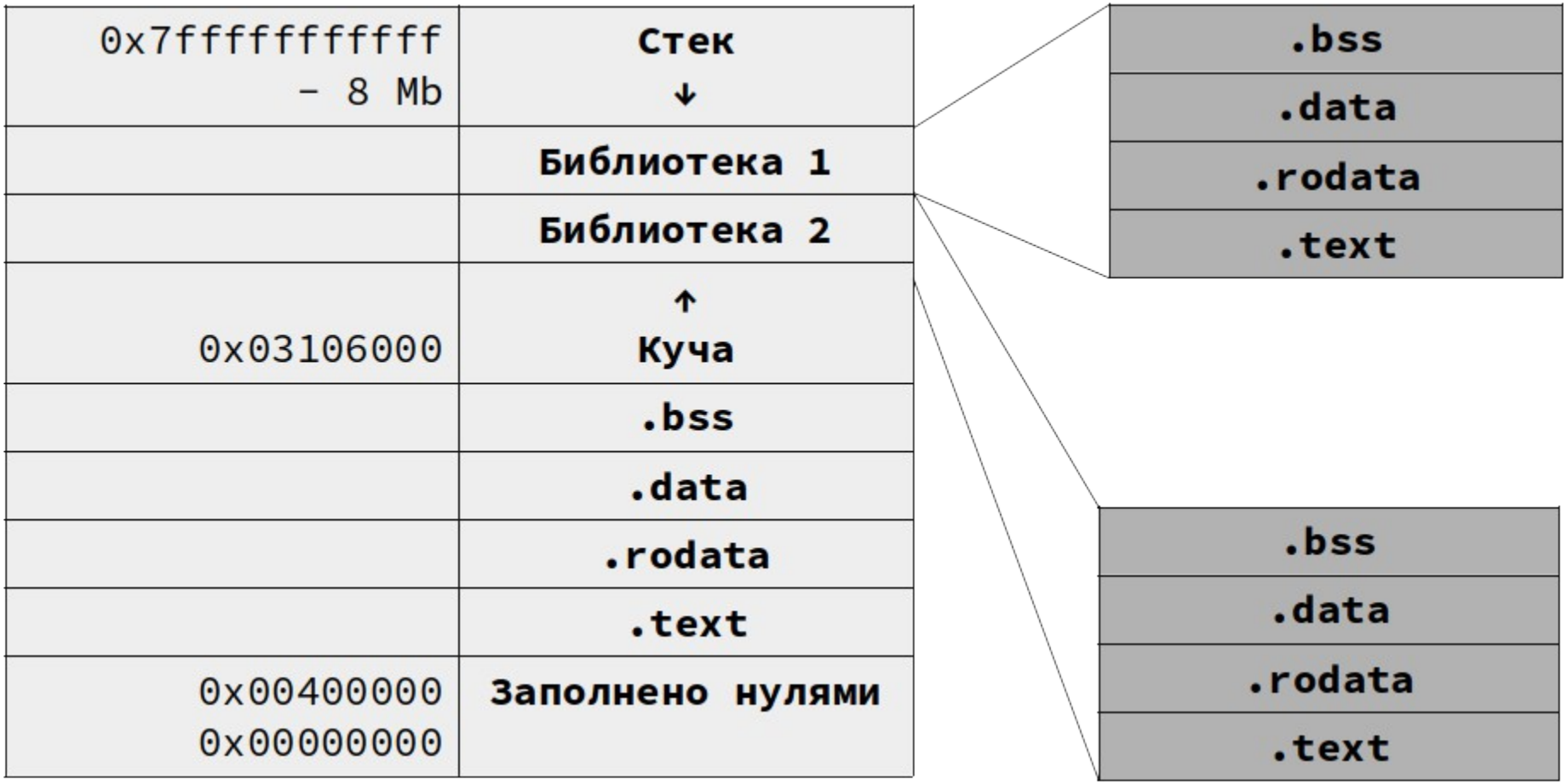
- **foo_bar.elf** - это программа, если существует **точка входа** по определенному адресу

Для Linux x86:
- по умолчанию `0x804800`
- настраивается
`-Wl,-Ttext-segment=`$ADDR$

- **foo_bar.elf** - это динамически загружаемая библиотека, если существует **таблица символов**

`-Wl,--export-dymanic`
или
`-shared`

Размещение библиотек в памяти

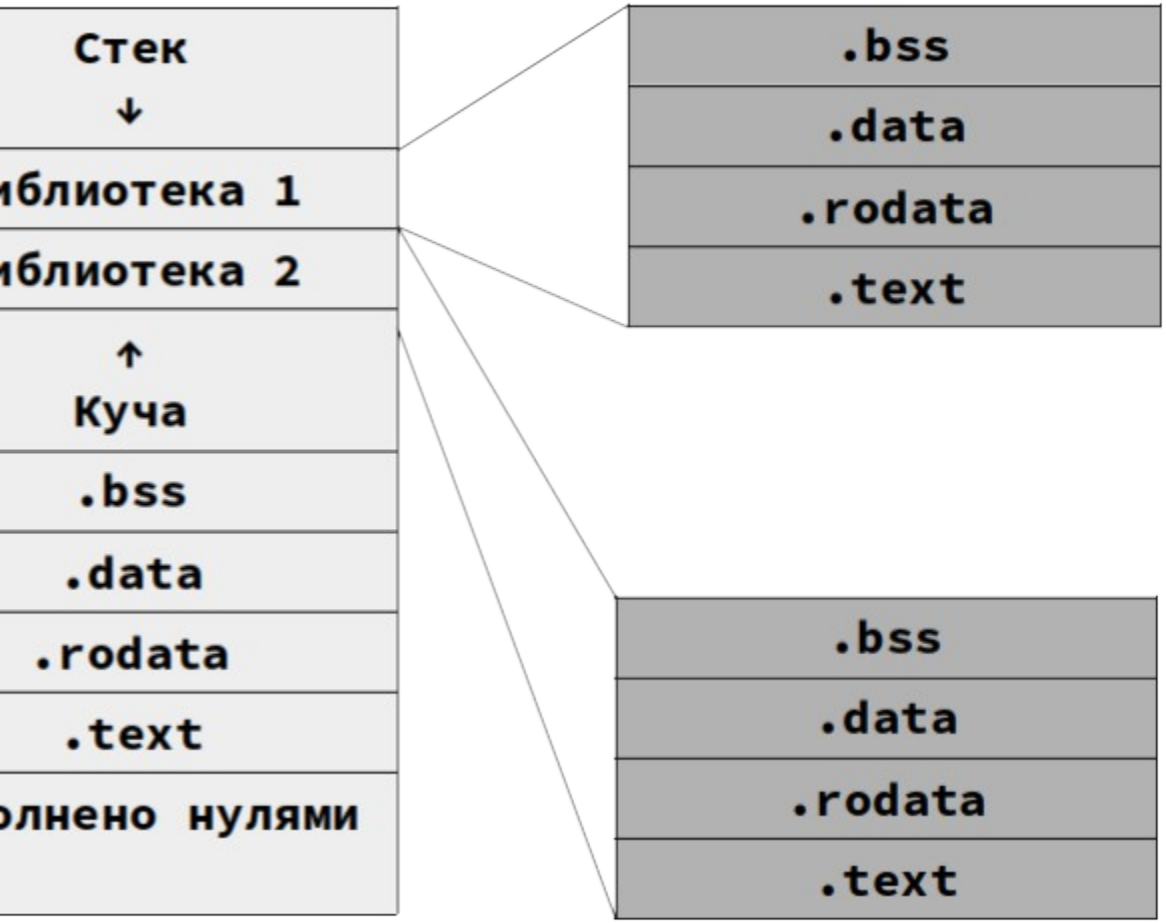


/lib[64]/ld-linux.so

1. Разместить файл в памяти
2. Загрузить библиотеку из переменной окружения LD_PRELOAD
3. Найти все необходимые библиотеки, построить граф зависимостей
4. Загрузить все библиотеки в пямять
5. Создать таблицы GOT и PLT
6. Передать управление программе на точку входа (функция _start)

Размещение библиотек в памяти

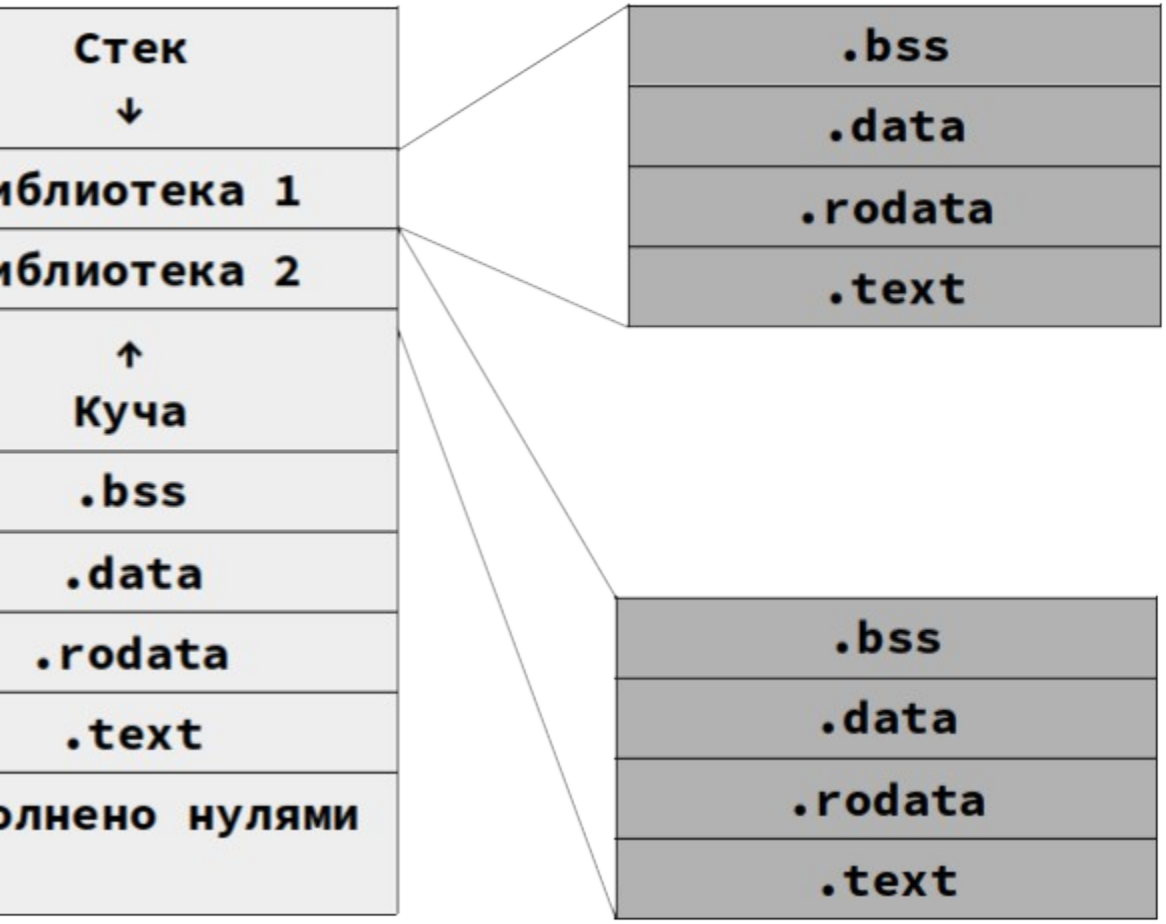
PLT – Procedure Linkage Table
GOT – Global Offset Table



foo_bar.elf

```
0x0010: int var = 10
0x0014: void foo1 ()
. . . .
0x00A0: void foo2 () {
        call foo1 + @PLT
0x00A2: ret }
0x00A4: void bar () {
        alloca loc
0x00A6: loc = *(&var+@GOT)
0x00A8: call foo2 + @PLT
0x00AA: ret }
```

Размещение библиотек в памяти



foo_bar.elf

```
0x0010: int var = 10
0x0014: void foo1 ()
. . . .
0x00A0: void foo2 () {
        call foo1 # 0x0014
0x00A2: ret }
0x00A4: void bar () {
        alloca loc
0x00A6: loc = var # 0x0010
0x00A8: call foo2 # 0x00A0
0x00AA: ret }
```

Опции gcc/clang

- **`-fPIC`** - скомпилировать исходный текст в позиционно-независимый код, который может быть использован в библиотеках
- **`-fPIE`** - аналогично **`-fPIC`**, но код также может быть использован для исполняемых файлов
- **`-pie`** - скомпоновать позиционно-независимый выполняемый файл

Position-Independent Executable

- Файл одновременно может быть использован как программа, так и библиотека
- Ядро ОС может размещать файл случайным образом (ASLR)

`ptrace` - см. пример про фильтрацию

- Можно запустить программу под ptrace
- Можно подключиться через PTRACE_ATTACH

- Нельзя исследовать SUID-программы
- Можно исследовать только те, программы, которые запущены из под того же пользователя

(тем не менее, планета в опасности!)

Address Space Layout Randomization

- OpenBSD - используется по умолчанию
- Linux начиная с версии 2.6.12 - используется, если программа PIE

PIC/PIE в мирных целях

- Разделяемые библиотеки - больше возможностей по размещению в адресном пространстве
- Возможность подгружать библиотеки и выгружать из во время выполнения программы - **plugin'ы**

Что такое plugin

- Самодостаточный набор функций и классов
- Может быть отдельно протестирован
- Может быть реализован другой командой разработчиков или на другом языке программирования, либо вообще не иметь исходных текстов

Механизм dlopen/dlsym

```
void *dlopen(const char *lib_name, int flags)
HMODULE LoadLibraryA(LPCSTR lib_name, DWORD flags)
```
--- *загрузить библиотеку*

```
void *dlsym(void *lib, cons char *func_name)
FARPROC GetProcAddress(HMODULE lib, LPCSTR func_name)
```
--- *найти функцию в библиотеке*

Механизм dlopen/dlsym

```
#include <dlfcn.h>
void some_func() {
    void* lib =
       dlopen(
          "libSDL.so",
          RTLD_LAZY
       ); // Check for NULL!!!

    void* func_ptr =
       dlsym(lib, "SDL_Init");
          // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

```
#include <Windows.h>
void some_func() {
    HMODULE lib =
       LoadLibraryA(
          "winhttp.dll"
       ); // Check for NULL!!!

    FARPROC func_ptr =
       GetProcAddress(
         lib, "WinHttpConnect"
       ); // Check for NULL!!!

    (*func_ptr)(....); // Call
}
```

Вызов функции по имени

```
def func():
    print("Hello, World!");

func_name = input("Please enter func name: ")

func_ref = globals()[func_name]

func_ref()  # Hello, World!
```

Вызов функции по имени

```c
#include <dlfcn.h>

void callable() {}

void some_func() {
    void* lib =
       dlopen(NULL, 0);

    void* func_ptr =
       dlsym(lib, "callable");
          // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

```c
#include <Windows.h>

__declspec(dllexport)
void callable() {}

void some_func() {
    HMODULE lib =
       LoadLibrary(NULL);

    FARPROC func_ptr =
       GetProcAddress(
         lib, "callable"
       ); // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

https://docs.python.org/3/library/ctypes.html

- общая идея — использовать dlopen/LoadLibrary для загрузки произвольных библиотек во время выполнения
- доступ к функциям по Си-имени
- сигнатуры НЕ контролируются

Плагины на C++

- Доступен интерфейс в виде .h-файла
- Достаточно реализовать только одну внешнюю функцию, которая создаёт экземпляр класса на куче, и возвращает указатель на него
- В секции .rodata может храниться дополнительная метаинформация о плагине (например, в виде char[])

Пример реализации - механизм QtPlugin в фреймворке Qt [www.qt.io]