

Лекция 16

Исполняемые файлы

Исполняемый файл (executable)

- Файл, основное содержимое (payload) которого – программа для процессора
- После загрузки в память и первоначальной настройки исполняется непосредственно процессором, не требует программы-”интерпретатора”
- Тексты программ на python, bash; .jar-файлы (байт-код java); .net MSIL – не будем считать исполняемыми файлами

Native executable files

- У каждой операционной системы свой формат исполняемых файлов:
 - Windows: PE (portable executable)
 - MacOS, iOS: Mach-O (Mach Object)
 - Unix-like (Linux, *BSD, Solaris): ELF (executable and linkable format)
- Один и тот же формат может использоваться для разных процессоров
- Исполняемый файл специфичен для пары процессор-ОС (платформа)

Portability

- Операционные системы могут поддерживать “чужие” исполняемые файлы, если хост-система работает на том же процессоре.
 - FreeBSD поддерживает возможность запуска ELF-файлов для Linux той же процессорной архитектуры
 - Wine – поддержка запуска Windows-приложений на Unix-like системах

Инструменты работы

- Binutils – набор утилит для работы с исполняемыми файлами:
 - Objdump – отобразить содержимое:
 - objdump -d FILE – дизассемблировать
 - objdump -x FILE – вывести всю служебную инфу
 - objdump -j SECTION -s FILE – вывести содержимое секции SECTION в файле FILE
 - Readelf – анализ ELF-файла
 - Nm - список “символов” (symbols, т.е. именованных сущностей) в файле
- GDB, IDA PRO, OllyDBG - отладчики

ELF

- ELF – Executable and Linkable Format
- Применяется на многих Unix-like OS (Linux, *BSD, Solaris, ...)
- Для всех процессоров, на которых эти ОС работают
- Формат: объектных файлов (.o), статических библиотек (.a), динамических библиотек (.so), исполняемых файлов, дампов памяти, модулей ядра ОС (.ko)...

Структура ELF-файла

- Заголовок ELF-файла (magic bytes и структура Elf32_Ehdr)
- Таблица заголовков программы (program headers) – структура Elf32_Phdr
- Payload – содержимое файла
- Таблица заголовков секций (section headers) – структура Elf32_Shdr

(почти) минимальный ELF

- Заголовок файла – 52 байта
- 1 запись program header по 32 байта - 32
- Исполняемый код – 7
- Таблица имен секций – 17
- 3 записи секций по 40 байт – 120 байт
- Итог: 228 байт.
- Задание – получить такой ELF-файл

Program Headers

- Информация о том, как исполняемый файл должен грузиться в память
- LOAD – заданная область файла должна быть загружена в память
- LOAD требует выравнивания (align) в 0x1000 (4096 байт) – это размер страницы x86/x64 (рассмотрим далее)
- LOAD с флагами 'R E' – это сегмент кода
- LOAD с флагами 'RW' – это сегмент данных
- Записи Program Headers могут описывать накладывающиеся адреса

Section Headers

- Секция – это структурная единица в исполняемом файле
- Секции группируются в сегменты
- Типичные секции:
 - .text, .data, .bss, .rodata
 - .plt – procedure linkage table
 - .got – global offset table
 - .got.plt – global offset table for PLT

Динамическая компоновка

- Как правило (по умолчанию) программа, загружаемая на выполнение, не содержит в своем файле кода используемых библиотек
- Перед запуском программы требуется подгрузить динамические библиотеки
- Программа-загрузчик (интерпретатор) прописывается в секции `.interp` файла (для x86 - `/lib/ld-linux.so.2`)
- Операционная система загружает исполняемый файл, загружает требуемый загрузчик и запускает его
- Загрузчик выполняет подгрузку библиотек и передает управление программе

Динамические символы

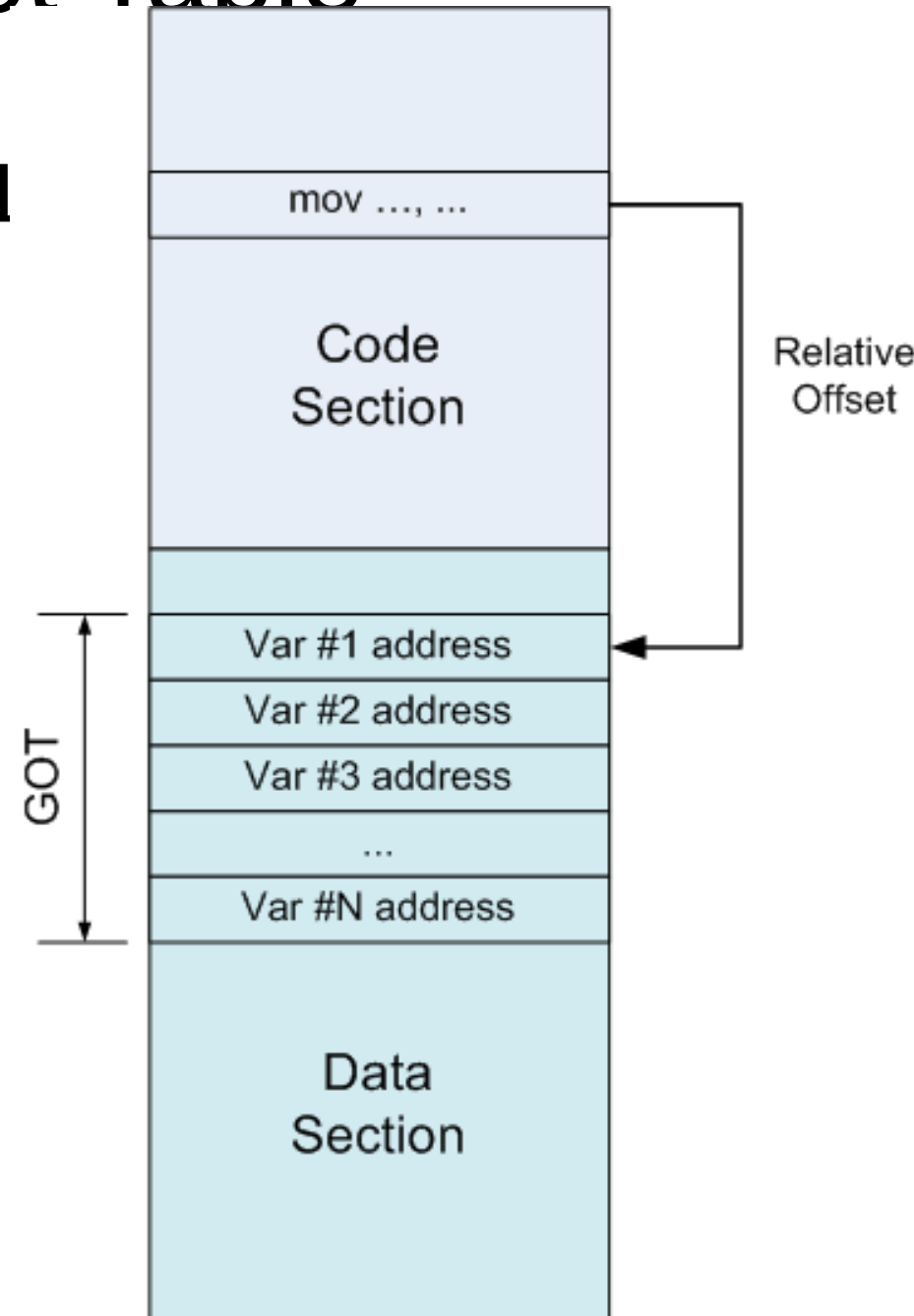
- Секция `.dynsym` содержит таблицу динамических символов, то есть имен, требуемых из данного файла или предоставляемых данным файлом
- Секция `.rel.dyn` содержит таблицу перемещений для переменных, т. е. адреса ячеек памяти, которые должны быть изменены
- Секция `.rel.plt` содержит таблицу перемещений для внешних функций

Global Offset Table

- Чтобы не модифицировать код программы, все перемещения находятся в специальных секциях
 - .got – для переменных
 - .got.plt – для функций
- После работы загрузчика таблица .got содержит абсолютные адреса переменных

Global Offset Table

- `_GLOBAL_OFFSET_TABLE`
- `L@GOT`



Использование GOT

- `_GLOBAL_OFFSET_TABLE_` - это смещение относительно **адреса текущей инструкции** до адреса GOT
- `LABEL@GOT` - это смещение относительно GOT до ячейки памяти, в которой хранится правильный адрес, по которому размещается LABEL

Procedure Linkage Table (PLT)

- В динамически скомпонованных программах часть GOT отводится под хранение адресов функций из динамических библиотек
- Если `printf` – функция из динамической библиотеки, то `call printf` заменяется на `call printf@plt`
- `printf@plt` - специальная функция (трамплин) для передачи управления в библиотеку

Procedure linkage table (PLT)

080483f0 <dynlink>:

80483f0:	ff 35 04 a0 04 08	pushl	0x804a004
80483f6:	ff 25 08 a0 04 08	jmp	*0x804a008
80483fc:	00 00	add	%al, (%eax)

08048410 <printf@plt>:

8048410:	ff 25 10 a0 04 08	jmp	*0x804a010
8048416:	68 08 00 00 00	push	\$0x8
804841b:	e9 d0 ff ff ff	jmp	80483f0 <dynlink>

0804a000 <_GLOBAL_OFFSET_TABLE_>:

804a000:	14 9f 04 08	.int	0x8049f14 <_DYNAMIC>
804a004:	00 00 00 00	.int	0
804a008:	00 00 00 00	.int	0
804a00c:	06 84 04 08	.int	0x8048406
804a010:	16 84 04 08	.int	0x8048416

Lazy binding

- При первом вызове `<printf@plt>` управление попадет в динамический загрузчик. В стеке будет передано смещение на дескриптор загружаемой функции
- Динамический загрузчик запишет в GOT адрес функции `printf` в загруженной динамической библиотеке
- Все последующие вызовы будут передавать управление сразу на `printf` в динамической библиотеке

Зачем нужны trampлины

- Компилятор и ассемблер не имеют информацию, в каком режиме будет компоноваться программа
 - они всегда генерируют инструкцию `CALL FUNC` для вызова функции `FUNC`
 - Компоновщик (linker) при необходимости заменит `CALL FUNC` на `CALL FUNC@PLT` и сгенерирует трамплин **`FUNC@PLT`**
- Если в Си-коде берется адрес функции, например, `&printf`, то компоновщик заменяет на **`&printf@plt`** без модификации машинного кода
- Позволяют реализовывать ленивое связывание (lazy binding)

Ссылки

- <http://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-libraries>
- <https://www.cs.stevens.edu/~jschauma/810/elf.html>
- <https://ejudge.ru/study/3sem/elf.html>
-