Анализ кода и информационная безопасность

Лекция 10

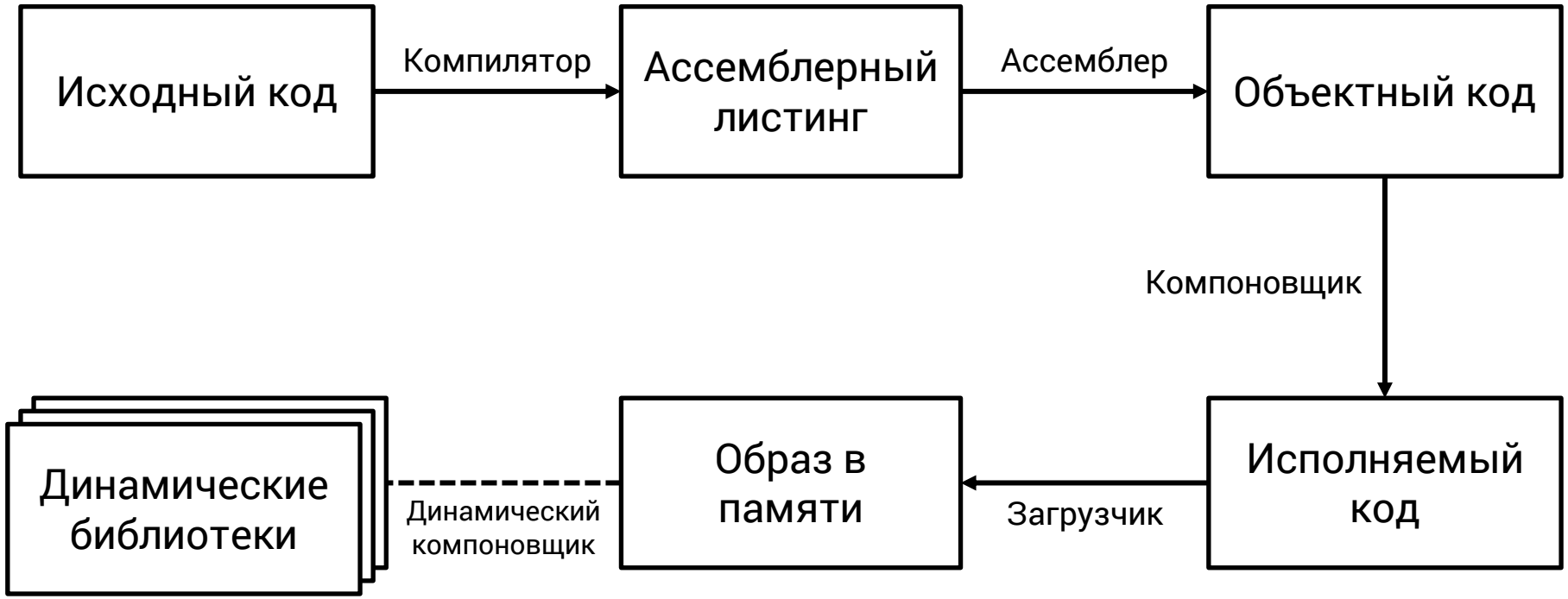
Статическое дизассемблирование

МГУ / ВМК / СП

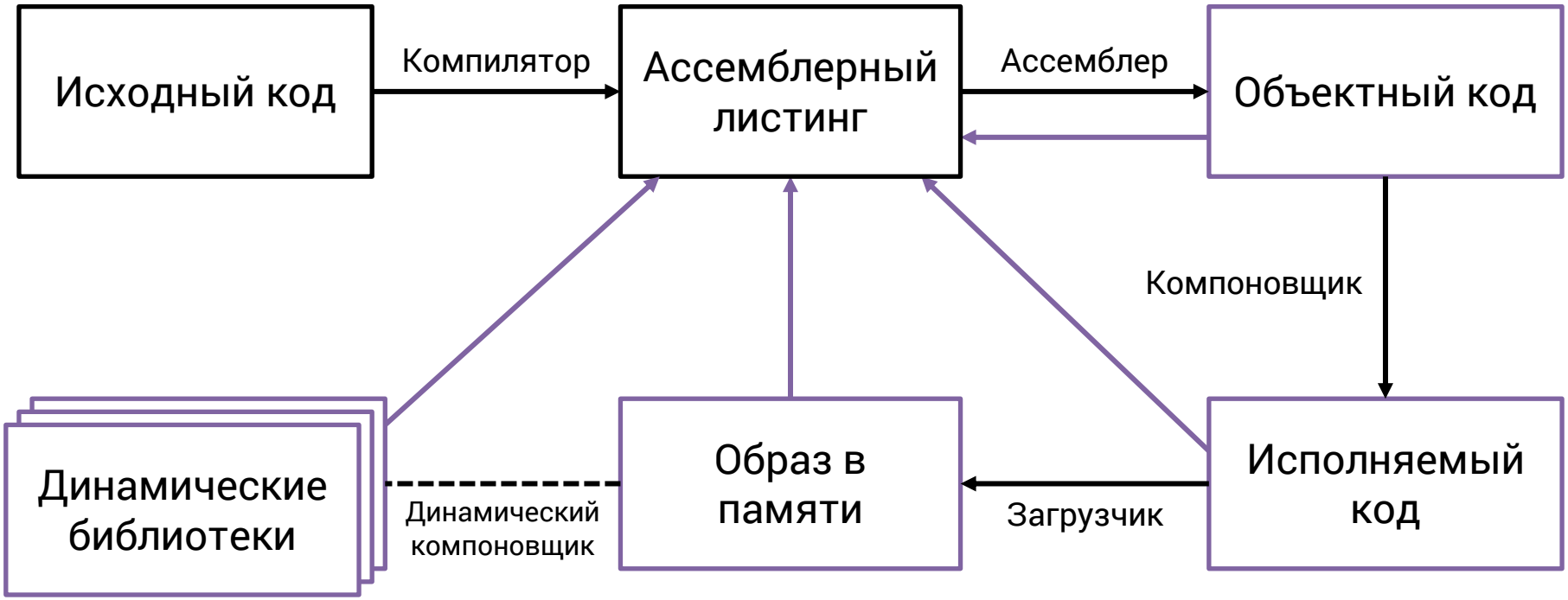
1001

СТАТИЧЕСКОЕ ДИЗАССЕМБЛИРОВАНИЕ

Задача дизассемблирования



Задача дизассемблирования



Задача дизассемблирования

- Вход — бинарный образ программы (или её составной части) в виде исполняемого/объектного файла/библиотеки (PE, ELF, Mach-O…) или в неструктурированном виде (снимок памяти процесса).
- Выход — ассемблерный листинг, содержащий секции кода и данных.

Разбиение на подзадачи:

a) разделение кода и данных;
b) выявление границ команд в секциях кода;
c) декодирование команд;
d) группировка и примитивная типизация в секциях данных;
e) переход от абсолютных адресов к символам;
f) формирование текста ассемблерного листинга.

Разделение кода и данных
Архитектура фон Неймана

1. Принцип неразличимости команд и данных:
   - по значению машинного слова невозможно определить, что оно собой представляет — команду или данные.
2. Принцип линейности и однородности памяти:
   - в различные моменты времени одно и то же машинное слово может восприниматься и как команда, и как данные;
   - пример 1 — применение таблиц перемещения;
   - пример 2 — динамическая генерация кода.
3. Принцип хранимой программы.
4. Принцип автоматической работы.
5. Принцип последовательного выполнения.

Разделение кода и данных

Архитектура фон Неймана

В архитектуре фон Неймана кодом являются те и только те участки образа программы, из которых при её работе процессором могут выбираться команды.

Считается, что в такой постановке задача разделения кода и данных в общем случае эквивалентна задаче останова, т.е. алгоритмически неразрешима.

Выявление границ команд

- При наличии декодера машинных команд достаточно определить начало некоторой команды, тогда её границы могут быть вычислены путём декодирования.
- Если известен некоторый адрес, соответствующий коду, то можно произвести декодирование цепочки команд, начиная с этого адреса:
  - последовательный просмотр (linear sweep);
  - рекурсивный спуск (recursive descent).
- Источники знаний об адресах кода:
  - точка входа в программу;
  - структурные элементы формата исполняемого файла (таблицы символов, таблицы связывания);
  - результаты сигнатурного поиска;
  - интерактивная работа пользователя.

Последовательный просмотр

```
linear_sweep(code_start, code_end)
{
    // Цикл по всему сегменту кода.
    for (code_ptr = code_start; code_ptr != code_end; ) {
        // Декодировать очередную машинную команду.
        instruction = decode(code_ptr);

        // Выдать результат.
        yield instruction;

        // Перейти к следующей команде.
        code_ptr += instruction.size;
    }
}
```

Предполагается, что в сегменте кода встречается только код, а команды следуют друг за другом неразрывно до самого конца сегмента.

Верно ли это хотя бы для программ без защиты от анализа?

Рекурсивный спуск

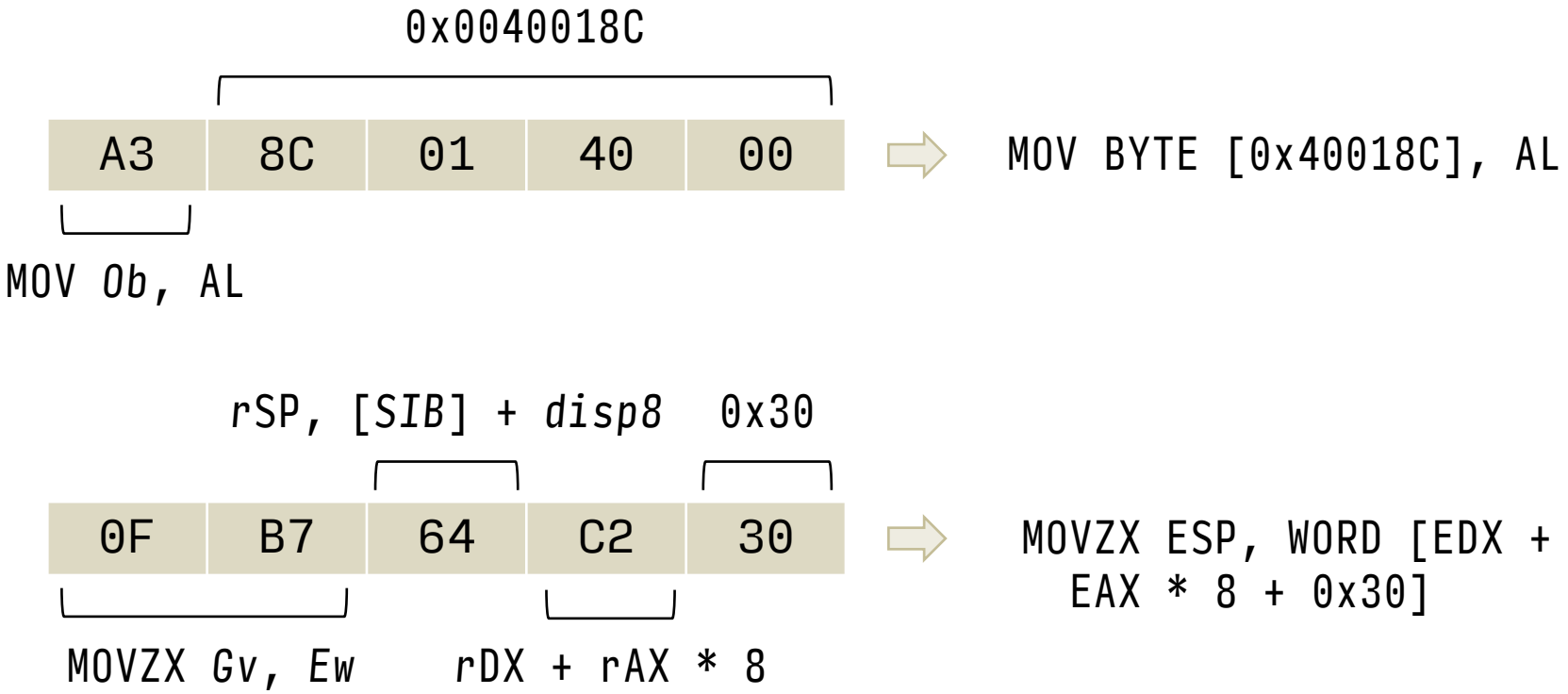
```
recursive_descent(code_start)
{
    // Создать очередь адресов, куда изначально заносится code_start.
    queue = Queue(code_start)

    // Цикл по элементам очереди.
    while (!is_empty(queue)) {
        // Выбрать очередной адрес кода из очереди.
        code_ptr = dequeue(queue);

        // Декодировать очередную машинную команду.
        instruction = decode(code_ptr);

        // Выдать результат.
        yield instruction;

        // Добавить в очередь все адреса, куда может перейти управление.
        for (target : instruction.targets) enqueue(queue, target);
    }
}
```

Рекурсивный спуск
Классификация команд

- С точки зрения набора адресов передачи управления:
  - команды, не являющиеся командами передачи управления;
  - команды безусловной передачи управления;
  - команды условной передачи управления;
  - команды вызова подпрограммы;
  - команды возврата из подпрограммы;
  - команды системного вызова (в т.ч. программные прерывания);
  - команды останова.
- С точки зрения вычислимости адреса перехода:
  - передача управления по абсолютному константному адресу;
  - передача управления по константному смещению относительно счётчика команд;
  - передача управления по вычисляемому адресу.

Декодирование

Декодирование — преобразование последовательности байтов в описание машинной команды.



Декодирование

- Во многих современных процессорах декодирование зависит от контекста.
- Контекст в x86 включает в себя два признака из флагов сегмента кода:
  - CS.L — признак 64-разрядного режима;
  - CS.D — признак размера данных по умолчанию (16 или 32 бита), совместно с CS.L не используется.
- Контекст в ARMv7 включает признак используемого набора команд:
  - «классический» ARM — 32-разрядные команды;
  - Thumb, Thumb-2 — команды могут быть 16- и 32-разрядными.
- В общем случае код команды имеет переменную длину.
- Для того, чтобы определить границы команды, необходимо её полностью декодировать с учётом контекста.

Декодирование
x86 и x86-64

- Разбор префиксов команды:
  - VEX;
  - REX;
  - legacy-префиксы: address size override, operand size override;
  - LOCK.
- Разбор кода операции и форматов операндов:
  - мнемоника;
  - для каждого операнда: режим адресации и размер.
- Разбор операндов.
- «Нормализация» представления команды:
  - `XCHG rAX, rAX` — `NOP`.

Декодирование
Библиотеки

- Capstone (ARMv7, ARMv8, M68K, MIPS, PPC, SPARC, SystemZ, XCore, x86, x86-64) — http://www.capstone-engine.org/;
- Intel XED (x86, x86-64) — https://software.intel.com/en-us/articles/xed-x86-encoder-decoder-software-library;
- diStorm3 (x86, x86-64) — https://github.com/gdabah/distorm;
- quix86 (x86, x86-64) — https://github.com/ispras/quix86.

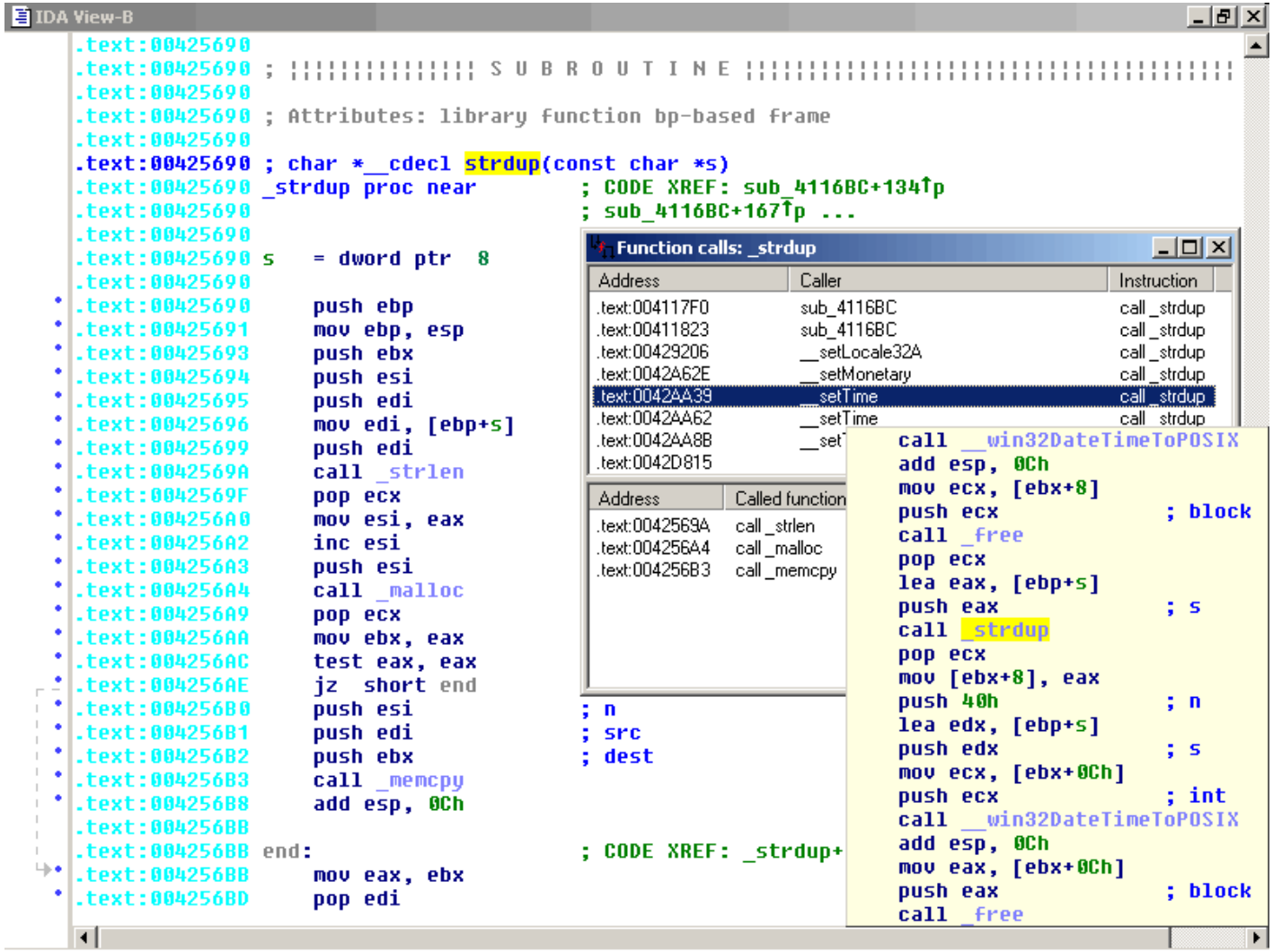
Группировка и типизация данных

- Под группировкой понимается объединение последовательных байтов в единую логическую единицу, например в 32-разрядное значение.
- Под примитивной типизацией понимается выбор одной из возможных эквивалентных записей ассемблерных директив описания данных:
  - `DD 0x6C6C6548, 0x6F77206F, 0x21646C72, 0x0000000A;`
  - `DB “Hello world!”, 10, 0, 0, 0.`
- Основной источник знаний о структуре данных программы — её код. Анализируются восстановленные команды программы на предмет доступа к соответствующему участку памяти.
- Пример: `FLD QWORD [A]` — скорее всего по адресу *A* расположено 64-разрядное число с плавающей точкой.
- Уточнения может вносить пользователь интерактивно.

Переход от абсолютных адресов к символам

- Всем константным адресам, которые фигурируют в командах (т.е. адресам переходов и адресам данных) сопоставляются метки.
- Имена меток (символы) выбираются одним из трёх способов:
  - генерируются дизассемблером (как правило, малоосмысленные цифровые номера);
  - берутся из таблиц символов (если они есть);
  - задаются пользователем.
- В общем случае после этого преобразования код не становится перемещаемым. — Почему?

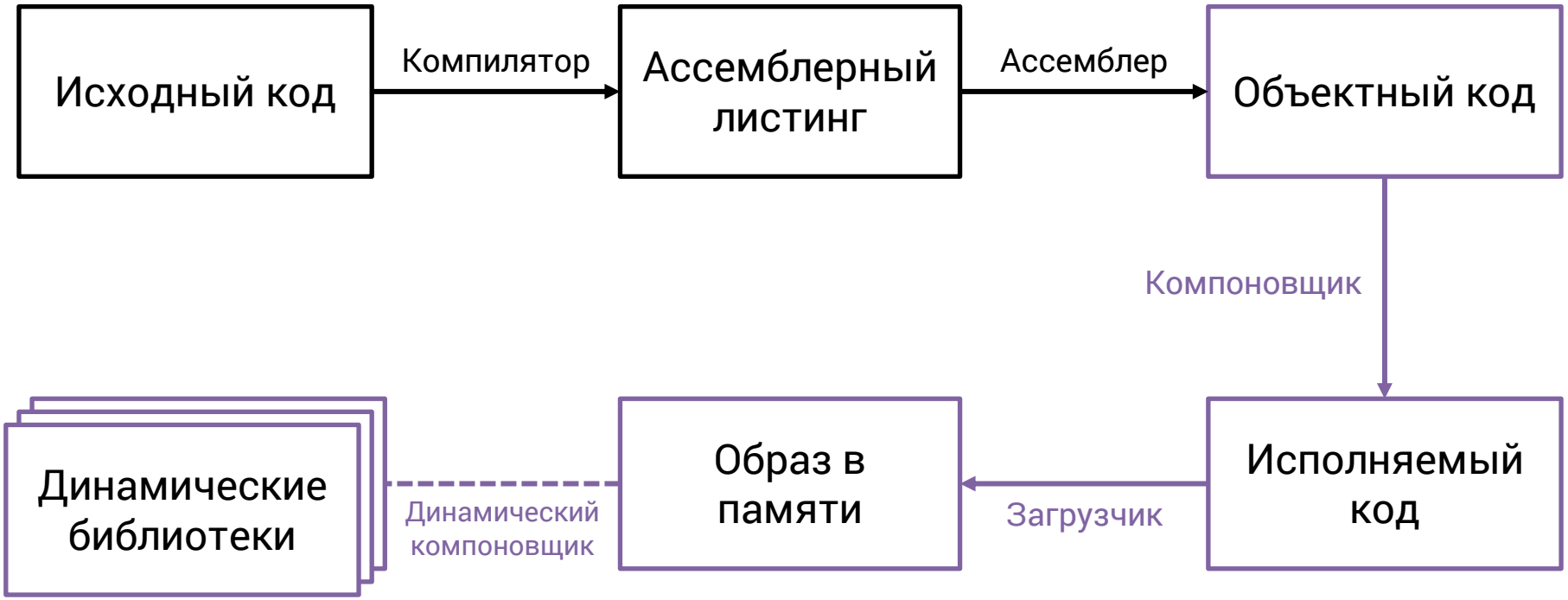
IDA Pro


IDA View-B
.text:00425690 ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:00425690 ; Attributes: library function bp-based frame
.text:00425690 ; char *__cdecl strdup(const char *s)
.text:00425690 _strdup proc near ; CODE XREF: sub_4116BC+134↑p
.text:00425690 ; sub_4116BC+167↑p ...
.text:00425690 s = dword ptr 8
.text:00425690 push ebp
.text:00425691 mov ebp, esp
.text:00425693 push ebx
.text:00425694 push esi
.text:00425695 push edi
.text:00425696 mov edi, [ebp+s]
.text:00425699 push edi
.text:0042569A call _strlen
.text:0042569F pop ecx
.text:004256A0 mov esi, eax
.text:004256A2 inc esi
.text:004256A3 push esi
.text:004256A4 call _malloc
.text:004256A9 pop ecx
.text:004256AA mov ebx, eax
.text:004256AC test eax, eax
.text:004256AE jz short end
.text:004256B0 push esi ; n
.text:004256B1 push edi ; src
.text:004256B2 push ebx ; dest
.text:004256B3 call _memcpy
.text:004256B8 add esp, 0Ch
.text:004256BB
.text:004256BB end: ; CODE XREF: _strdup+
.text:004256BB mov eax, ebx
.text:004256BD pop edi
Function calls: _strdup
Address Caller Instruction
.text:004117F0 sub_4116BC call _strdup
.text:00411823 sub_4116BC call _strdup
.text:00429206 __setLocale32A call _strdup
.text:0042A62E __setMonetary call _strdup
.text:0042AA39 __setTime call _strdup
.text:0042AA62 __setTime call _strdup
.text:0042AA8B __set
.text:0042D815
Address Called function
.text:0042569A call _strlen
.text:004256A4 call _malloc
.text:004256B3 call _memcpy
call __win32DateTimeToPOSIX
add esp, 0Ch
mov ecx, [ebx+8]
push ecx ; block
call _free
pop ecx
lea eax, [ebp+s]
push eax ; s
call _strdup
pop ecx
mov [ebx+8], eax
push 40h ; n
lea edx, [ebp+s]
push edx ; s
mov ecx, [ebx+0Ch]
push ecx ; int
call __win32DateTimeToPOSIX
add esp, 0Ch
mov eax, [ebx+0Ch]
push eax ; block
call _free

1002

ФОРМАТЫ ИСПОЛНЯЕМЫХ И ОБЪЕКТНЫХ ФАЙЛОВ

Исполняемые и объектные файлы



Классификация

- Объектные файлы:
  - подлежат дальнейшей статической компоновке;
  - могут иметь неразрешённые зависимости от других объектных файлов и библиотек;
  - объектам (функциям и данным) не назначены окончательные адреса.
- Исполняемые файлы и динамические библиотеки:
  - могут подвергаться дальнейшей динамической компоновке;
  - могут иметь неразрешённые зависимости только от динамических библиотек;
  - объектам (функциям и данным) назначены окончательные адреса за исключением возможного сдвига на константу при загрузке.

ELF

ELF (Executable and Linkable Format) — формат объектных, исполняемых файлов и динамических библиотек, используемый в UNIX-системах.

Спецификации:

- ELF32 — http://www.skyfree.org/linux/references/ELF_Format.pdf;
- ELF64 — https://www.uclibc.org/docs/elf-64-gen.pdf.

Для отдельных целевых платформ выпущены также так называемые “processor supplements,” описывающие дополнительные особенности на данных платформах.

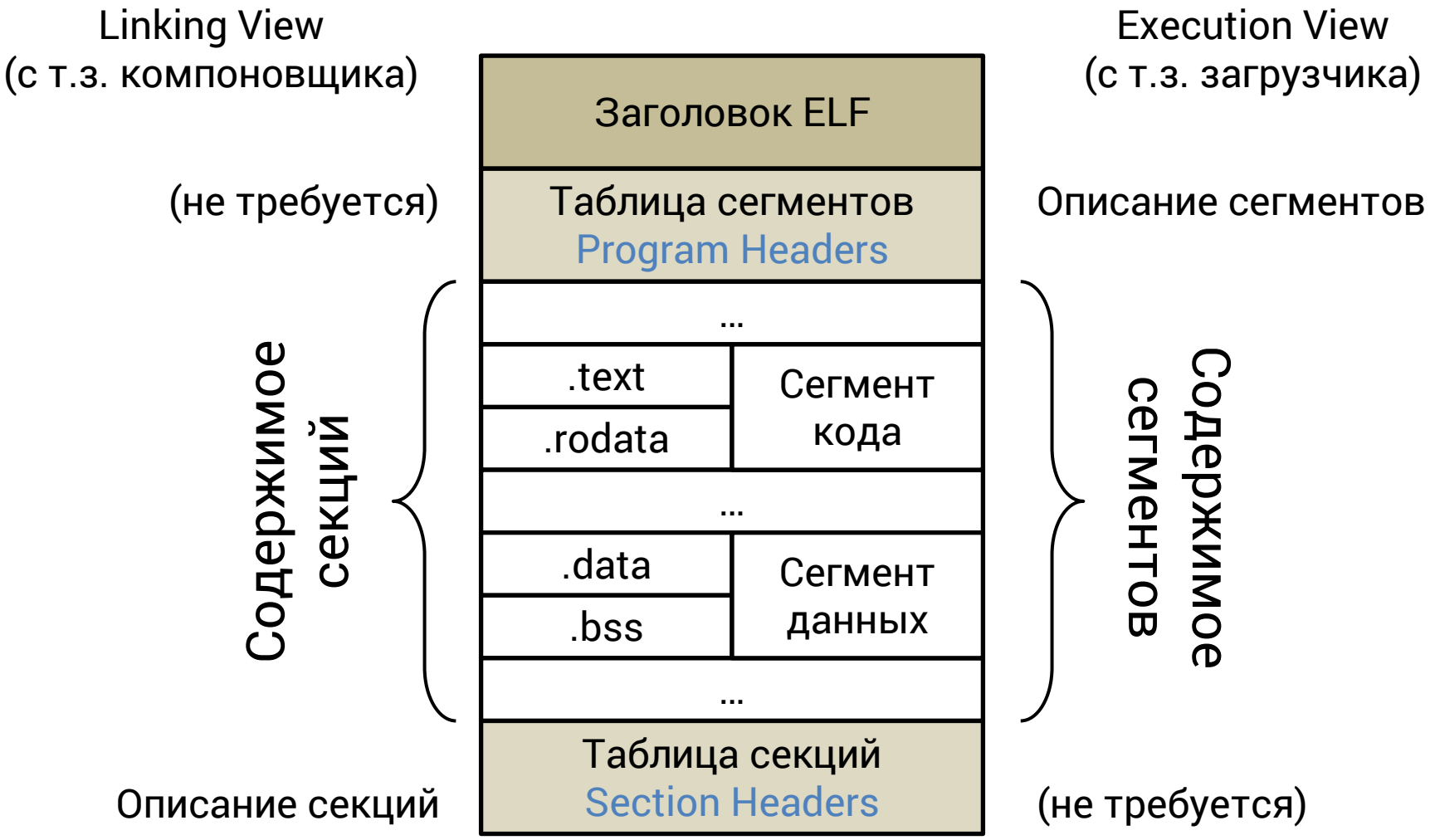
ELF
Типы файлов

1. Перемещаемые (REL):
   - идеологически являются объектными файлами;
   - объекты в них не имеют окончательных адресов (даже относительно друг друга);
   - имеются ссылки на внешние символы, подлежащие разрешению;
   - обрабатываются статическим компоновщиком для создания исполняемого файла или динамической библиотеки.
2. Исполняемые (EXEC):
   - являются входом для загрузчика, входящего в состав ОС;
   - объекты имеют окончательные виртуальные адреса;
   - все внешние ссылки, кроме ссылок на символы из динамических библиотек, разрешены.

ELF
Типы файлов

3. Разделяемые объекты (DYN):
   - идеологически являются динамическими библиотеками;
   - поступают на вход динамическому компоновщику во время загрузки использующей библиотеку программы;

ELF



ELF
Таблица секций

- Образ программы рассматривается как набор секций.
- Характеристики секции:
  - имя;
  - тип;
  - флаги;
  - начальный адрес в виртуальной памяти;
  - начальное смещение в файле;
  - размер секции;
  - минимальное выравнивание;
  - другие характеристики.

ELF
Типы и флаги секций

Типы секций:

- NULL — игнорируется;
- PROGBITS — объекты программы (код или данные);
- NOBITS — неинициализированные объекты программы (.bss);
- SYMTAB, DYNSYM — таблицы символов для компоновки;
- STRTAB — таблица имён секций, объектов, единиц трансляции;
- REL, RELA — таблицы перемещения (relocations);
- HASH — хеш-таблицы имён;
- DYNAMIC — таблицы для динамического компоновщика;
- NOTE — дополнительные сведения о программе.

Флаги секций:

- SHF_WRITE — секция содержит изменяемые данные;
- SHF_ALLOC — секция должна загружаться в память при выполнении;
- SHF_EXECINSTR — секция содержит машинные команды.

ELF

Пример таблицы секций

```
$ readelf -S hello.o
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000400238  00000238
       000000000000001c  0000000000000000   A       0     0     1
  [ 5] .dynsym           DYNSYM           00000000004002b8  000002b8
       0000000000000060  0000000000000018   A       6     1     8
  [ 9] .rela.dyn         RELA             0000000000400380  00000380
       0000000000000018  0000000000000018   A       5     0     8
  [10] .rela.plt         RELA             0000000000400398  00000398
       0000000000000030  0000000000000018  AI       5    24     8
  [12] .plt              PROGBITS         00000000004003f0  000003f0
       0000000000000030  0000000000000010  AX       0     0     16
  [13] .plt.got          PROGBITS         0000000000400420  00000420
       0000000000000008  0000000000000000  AX       0     0     8
  [14] .text             PROGBITS         0000000000400430  00000430
       0000000000000192  0000000000000000  AX       0     0     16
  [16] .rodata           PROGBITS         00000000004005d0  000005d0
       0000000000000011  0000000000000000   A       0     0     4
  [25] .data             PROGBITS         0000000000601028  00001028
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS           0000000000601038  00001038
       0000000000000008  0000000000000000  WA       0     0     1
```

ELF

Пример таблицы символов

```
$ readelf -s hello.o
Symbol table '.dynsym' contains 4 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND puts@GLIBC_2.2.5 (2)
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Symbol table '.symtab' contains 67 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
    28: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    29: 0000000000600e20     0 OBJECT  LOCAL  DEFAULT   21 __JCR_LIST__
    30: 0000000000400480     0 FUNC    LOCAL  DEFAULT   14 deregister_tm_clones
    31: 00000000004004c0     0 FUNC    LOCAL  DEFAULT   14 register_tm_clones
    32: 0000000000400500     0 FUNC    LOCAL  DEFAULT   14 __do_global_dtors_aux
    33: 0000000000601038     1 OBJECT  LOCAL  DEFAULT   26 completed.7585
    34: 0000000000600e18     0 OBJECT  LOCAL  DEFAULT   20 __do_global_dtors_aux_fin
    35: 0000000000400520     0 FUNC    LOCAL  DEFAULT   14 frame_dummy
    36: 0000000000600e10     0 OBJECT  LOCAL  DEFAULT   19 __frame_dummy_init_array_
    37: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS hello.c
    38: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    39: 0000000000400700     0 OBJECT  LOCAL  DEFAULT   18 __FRAME_END__
    40: 0000000000600e20     0 OBJECT  LOCAL  DEFAULT   21 __JCR_END__
```

ELF

Пример таблицы перемещения

```
$ readelf -r hello.o
Relocation section '.rela.dyn' at offset 0x380 contains 1 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000600ff8  000300000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0

Relocation section '.rela.plt' at offset 0x398 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000601018  000100000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000601020  000200000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
```

Таблицы перемещения указывают, как необходимо изменить код или данные при перемещении объектов.

Каждый элемент таблицы содержит смещение изменяемого поля, символ, относительно адреса которого необходимо преобразовать секцию, тип адресного выражения и его константный аргумент (RELA).

ELF
Статическая компоновка

- Объединение секций из отдельных объектных файлов:
  - назначение адресов объектам.
- Разрешение символов:
  - применение перемещений из таблиц REL, RELA.
- Генерация исполняемого файла или динамической библиотеки:
  - построение таблицы сегментов для загрузчика.

ELF
Таблица сегментов

- Образ программы рассматривается как набор сегментов.
- Характеристики сегмента:
  - тип;
  - начальное смещение в файле;
  - начальный адрес в виртуальной памяти;
  - начальный адрес в физической памяти (обычно не используется);
  - размер сегмента в файле;
  - размер сегмента в памяти;
  - флаги;
  - минимальное выравнивание.

ELF
Типы и флаги сегментов

Типы сегментов:

- NULL — игнорируется;
- LOAD — загружаемый сегмент;
- DYNAMIC — сегмент содержит информацию, необходимую для динамической компоновки;
- INTERP — сегмент содержит ссылку на интерпретатор;
- NOTE — сегмент содержит дополнительные сведения о программе (например, необходим ли ей исполнимый стек).

Флаги сегментов:

- PF_X — доступ на выполнение;
- PF_W — доступ на запись;
- PF_R — доступ на чтение.

ELF

Пример таблицы сегментов исполняемого файла

```
$ readelf -l hello
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000000704 0x0000000000000704  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000228 0x0000000000000230  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x00000000000005e4 0x00000000004005e4 0x00000000004005e4
                 0x0000000000000034 0x0000000000000034  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1
```

ELF

Пример таблицы сегментов разделяемого объекта

```
$ readelf -l hello.so
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000754 0x0000000000000754  R E    200000
  LOAD           0x0000000000000e00 0x0000000000200e00 0x0000000000200e00
                 0x0000000000000228 0x0000000000000230  RW     200000
  DYNAMIC        0x0000000000000e18 0x0000000000200e18 0x0000000000200e18
                 0x00000000000001c0 0x00000000000001c0  RW     8
  NOTE           0x00000000000001c8 0x00000000000001c8 0x00000000000001c8
                 0x0000000000000024 0x0000000000000024  R      4
  GNU_EH_FRAME   0x00000000000006d0 0x00000000000006d0 0x00000000000006d0
                 0x000000000000001c 0x000000000000001c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e00 0x0000000000200e00 0x0000000000200e00
                 0x0000000000000200 0x0000000000000200  R      1
```

ELF
Работа загрузчика

- Работа загрузчика в простом случае сводится к проходу по таблице сегментов и построению по этой таблице образа программы в памяти процесса.
- Загрузчик — компонент ядра. Логично минимизировать его объём, чтобы уменьшить поверхность атаки.
- Загрузчик в ядре не способен напрямую выполнять динамическую компоновку. Вместо этого используется механизм ELF-интерпретатора.
- Исполняемый файл может содержать сегмент INTERP, в котором указан путь к другому исполняемому файлу, которому необходимо передать управление для окончательного формирования образа программы (в пользовательском режиме).
- В современных UNIX-системах этот интерпретатор и выполняет динамическое связывание.

ELF, PIC, PIE

- PIC (Position-Independent Code) — такая форма машинного кода, которая позволяет исполнять его с произвольного адреса в памяти.
- В PIC-коде не может быть абсолютных адресов. Используется адресация либо относительно счётчика команд (на архитектурах, которые это поддерживают), либо базовый адрес поддерживается на одном из регистров общего назначения.
- В современных системах все динамические библиотеки содержат только PIC-код. Это позволяет загрузить лишь одну копию библиотеки в память, а в разных адресных пространствах назначать ей различные базовые виртуальные адреса по необходимости.
- PIE (Position-Independent Executable) — исполняемый файл, содержащий только PIC-код.
- С точки зрения ELF-формата, PIE-программа — это разделяемый объект.
- PIE-программы участвуют в рандомизации адресного пространства.

Организация PIC-кода: GOT

GOT (Global Object Table) — таблица ссылок на внешние данные.

Когда в программе встречается ссылка на внешние данные, в ELF-файле должно появиться перемещение. Так как адрес кодируется как часть самой машинной команды, применение перемещения должно изменить код, а значит он не может быть перемещаемым.

Решение: вынести все такие ссылки в отдельную таблицу, и выполнять перемещения только в ней. Так как эта таблица существует отдельно от кода программы, применение к ней перемещений в каждом адресном пространстве не нарушает требований к PIC-коду.

Адресация самой таблицы при этом относительная.

Организация PIC-кода: PLT

PLT (Program Linkage Table) — таблица ссылок на внешние функции.

Аналогично GOT, собираем все внешние ссылки в отдельную таблицу, генерируя вместо реальных вызовов функций заглушки (stubs).

```
00000000004003f0 <PLT0>:
4003F0: PUSH   QWORD PTR [RIP + 0x200C12]    # 601008 <_GLOBAL_OFFSET_TABLE_ + 0x08>
4003F6: JMP    QWORD PTR [RIP + 0x200C14]    # 601010 <_GLOBAL_OFFSET_TABLE_ + 0x10>
4003FC: NOP    DWORD PTR [RAX + 0]

0000000000400400 <puts@plt>:
400400: JMP    QWORD PTR [RIP + 0x200C12]    # 601018 <_GLOBAL_OFFSET_TABLE_ + 0x18>
400406: PUSH   0
40040B: JMP    4003F0 <PLT0>

0000000000400410 <__libc_start_main@plt>:
400410: JMP    QWORD PTR [RIP + 0x200C0A]    # 601020 <_GLOBAL_OFFSET_TABLE_ + 0x20>
400416: PUSH   1
40041B: JMP    4003F0 <PLT0>
```

Организация PIC-кода: PLT

- При первом переходе на заглушку в соответствующей ячейке GOT записана ссылка на следующую команду заглушки (PUSH).
- Выполняемые команды подготавливают на стеке контекст, который описывает, на какую функцию необходимо передать управление, и вызывает динамический компоновщик.
- Компоновщик определяет адрес, соответствующий функции, и записывает его в соответствующую ячейку GOT.
- Компоновщик передаёт управление на целевую функцию.
- Все дальнейшие вызовы через данную ячейку таблицы PLT идут уже напрямую, минуя компоновщик.
- Описанная процедура называется ленивым связыванием.
- Существует также возможность выполнить связывание всех PLT-функций при старте программы. Зачем?

PE/COFF

Portable Executable (PE) — формат исполняемых файлов и динамических библиотек, используемый в Windows и UEFI.

Common Object File Format (COFF) — формат объектных файлов, используемый в Windows и UEFI.

Спецификация:
http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx.

PE

- Структурно PE-файл состоит из заголовка, таблицы секций, таблицы символов, нескольких директорий и содержимого секций.
- В отличие от ELF, в PE все адреса в файле — относительные (RVA).
- Директории включают в том числе следующие таблицы:
  - таблица экспорта;
  - таблица импорта;
  - таблица ресурсов;
  - таблица перемещения;
  - таблица сертификатов;
  - таблица отладочных данных;
  - таблица отложенного импорта;
  - таблица для .NET-приложений.

PE
Основные отличия от ELF

- Нет прямой возможности организации PIC-кода. Вместо этого DLL (и EXE, участвующие в механизме ASLR) перемещаются каждый раз, когда это необходимо. Отдельные страницы, в которых нет перемещений, могут использоваться одновременно в нескольких адресных пространствах.
- Особенности механизма импорта/экспорта:
  - возможность связывания как по имени, так и по порядковому номеру (ordinal);
  - возможность “forwarding”: одновременный экспорт и импорт некоторой функции (с переименованием);
  - по умолчанию связывание происходит при старте процесса, однако отдельная таблица может использоваться для ленивого связывания.

Литература к лекции

Основные источники

1. Касперски К., Рокко Е. Искусство дизассемблирования // СПб.: БХВ-Петербург. – 2008. — 896 с.
2. Eagle C. The IDA pro book: the unofficial guide to the world's most popular disassembler. – No Starch Press, 2011. — 676 с.
3. Vigna G. Static disassembly and code analysis // Malware Detection. – Springer US, 2007. – С. 19-41.
4. Executable and Linkable Format (ELF).
5. ELF-64 Object File Format.
6. Drepper U. How To Write Shared Libraries. — 2011.
7. Microsoft Portable Executable and Common Object File Format Specification.
8. Albertini A. PE 101: A Windows Executable Walkthrough.

Литература к лекции

Дополнительные источники

1. Дизассемблер Hex-Rays IDA Pro.
2. Библиотека декодирования Capstone.
3. Библиотека декодирования Intel XED.
4. Библиотека декодирования diStorm3.
5. Библиотека декодирования quix86.