Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе
Курс: Системное программирование
Тема: Утилита nm

Выполнил студент группы 13541/3 _____ Д.В. Круминьш
(подпись)

Преподаватель _____ Е.В. Душутина
(подпись)

Санкт-Петербург
2017 г.

Содержание

Постановка задачи

В данной работы необходимо:

- ознакомиться с работой утилиты nm;
- выполнение утилиты с различными ключами;
- провести анализ исходного кода утилиты, общий порядок функционирования;
- взаимосвязи, трассировка вызовов;
- провести модификацию программы.

Введение

nm — утилита, печатающая информацию о бинарных файлах(объектных файлах, библиотеках, исполняемых файлах и т. д.), прежде всего таблицу имён.

Является частью пакета Binutils проекта GNU.

В данной работе, рассматривается исходный код утилиты пакета Binutils версии 2.29(от 2017-07-24). Код утилиты nm, написан на языке Си и занимает 1811 строк.

Все опыты проводятся на виртуальной системе(64 битной), сведения о которой приведены далее.

```
root@kali:~/Desktop/testFolder# cat /etc/*release*
DISTRIB_ID=Kali
DISTRIB_RELEASE=kali-rolling
DISTRIB_CODENAME=kali-rolling
DISTRIB_DESCRIPTION="Kali GNU/Linux Rolling"
PRETTY_NAME="Kali GNU/Linux Rolling"
NAME="Kali GNU/Linux"
ID=kali
VERSION="2017.2"
VERSION_ID="2017.2"
ID_LIKE=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.kali.org/"
SUPPORT_URL="http://forums.kali.org/"
BUG_REPORT_URL="http://bugs.kali.org/"
```

Листинг 2.1: Информация о системе

```
root@kali:~/Desktop/testFolder# nm -V
GNU nm (GNU Binutils for Debian) 2.29
Copyright (C) 2017 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
```

Листинг 2.2: Версия утилиты

Обзор интерфейса утилиты

Была написана следующая программа.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    return 0;
6  }
```

Листинг 3.1: example.cpp

Далее данная программа была скомпилирована в объектный файл(example.out), имя которого, было передано как аргумент утилиты nm.

```
1  root@kali:~/Desktop/testFolder# gcc example.cpp –o example.out
2  root@kali:~/Desktop/testFolder# nm example.out
3  0000000000200e28 d _DYNAMIC
4  0000000000201000 d _GLOBAL_OFFSET_TABLE_
5  0000000000000690 R _IO_stdin_used
6                   w _ITM_deregisterTMCloneTable
7                   w _ITM_registerTMCloneTable
8  00000000000007d4 r __FRAME_END__
9  0000000000000694 r __GNU_EH_FRAME_HDR
10 0000000000201028 D __TMC_END__
11 0000000000201028 B __bss_start
12                  w __cxa_finalize@@GLIBC_2.2.5
13 0000000000201018 D __data_start
14 00000000000005b0 t __do_global_dtors_aux
15 0000000000200e20 t __do_global_dtors_aux_fini_array_entry
16 0000000000201020 D __dso_handle
17 0000000000200e18 t __frame_dummy_init_array_entry
18                  w __gmon_start__
19 0000000000200e20 t __init_array_end
20 0000000000200e18 t __init_array_start
21 0000000000000680 T __libc_csu_fini
22 0000000000000610 T __libc_csu_init
23                  U __libc_start_main@@GLIBC_2.2.5
24 0000000000201028 D _edata
25 0000000000201030 B _end
26 0000000000000684 T _fini
27 00000000000004b8 T _init
28 00000000000004f0 T _start
29 0000000000201028 b completed.7001
30 0000000000201018 W data_start
31 0000000000000520 t deregister_tm_clones
32 00000000000005f0 t frame_dummy
33 00000000000005fa T main
34 0000000000000560 t register_tm_clones
```

Листинг 3.2: Вывод утилиты nm

В выводе для каждого символа показано:

- Значение символа, в системе исчисления выбранной опциями (в зависимости от аргумента), или в шестнадцатеричной по умолчанию.
- Тип символа. Если символ написан маленькими буквами, то он локальный. Иначе он глобальный (внешний).
- Имя символа.[1]

Подробный разбор вывода утилиты, приведен в примерах работы.

3.1 Объектный файл

Прежде чем разбирать интерфейс программы, разберем что из себя представляет объектный файл.

С помощью утилиты **file** узнаем какой тип имеет полученный объектный файл.

```
1 root@kali:~/Desktop/testFolder# file example.out
2 example.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
    ↪ dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/
    ↪ Linux 2.6.32, BuildID[sha1]=6401ad5a9bdd30468dff292096d0ab9ea3ad38a5, not
    ↪  stripped
```

Листинг 3.3: Вывод утилиты file

ELF(Executable and Linking Format)(Формат Исполняемых и Связываемых файлов). Это одна из разновидностей форматов для исполняемых и объектных файлов, используемых в UNIX-системах и не только.

Заголовок файла (ELF Header) имеет фиксированное расположение в начале файла и содержит общее описание структуры файла и его основные характеристики, такие как: тип, версия формата, архитектура процессора, виртуальный адрес точки входа, размеры и смещения остальных частей файла.

Файлы ELF имеют два типа разбиений. Программный заголовок (program header) соответствует сегментам, которые будут использованы при исполнении. Заголовок секций (section header) перечисляет секции исполняемого файла. [2]

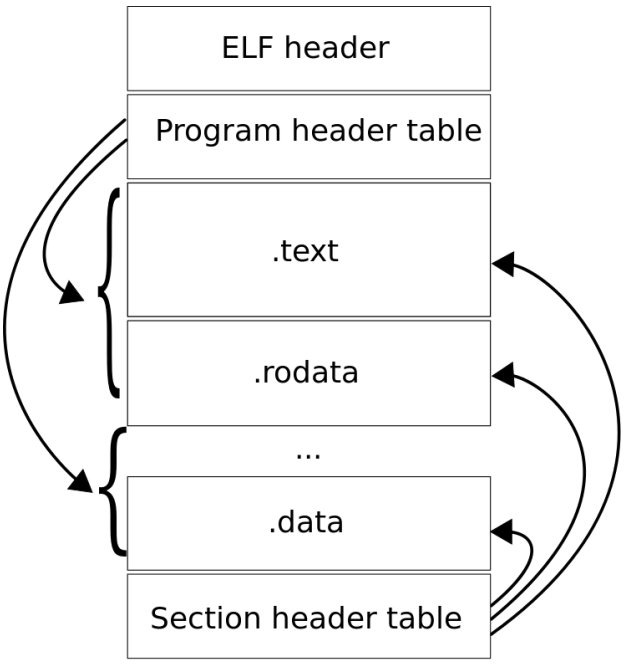


Рис. 3.1: Структура elf файла

Рассмотрим содержимое **ELF header**, с помощью команды **readelf -h**.

```
1 root@kali:~/Desktop/testFolder# readelf -h example.out
2 ELF Header:
3   Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
4   Class:                             ELF64
```

```
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x4f0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          6416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         29
  Section header string table index: 28
```

Листинг 3.4: Вывод содержимого ELF header

В данном листинге обратим внимание на значение **Entry point address**, которое соответствует значению **0x4f0**. Если вернуться к листингу 3.2, то в строчке 28 для данного адреса указан символ **_start**, точка входа в программу.

Рассмотрим что содержит в себе **SHT**(Section Header Table).

```
root@kali:~/Desktop/testFolder# readelf -S example.out
There are 29 section headers, starting at offset 0x1910:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000000238  00000238
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE             0000000000000254  00000254
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .note.gnu.build-i NOTE             0000000000000274  00000274
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH         0000000000000298  00000298
       000000000000001c  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM           00000000000002b8  000002b8
       0000000000000090  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB           0000000000000348  00000348
       000000000000007d  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM           00000000000003c6  000003c6
       000000000000000c  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED          00000000000003d8  000003d8
       0000000000000020  0000000000000000   A       6     1     8
  [ 9] .rela.dyn         RELA             00000000000003f8  000003f8
       00000000000000c0  0000000000000018   A       5     0     8
  [10] .init             PROGBITS         00000000000004b8  000004b8
       0000000000000017  0000000000000000  AX       0     0     4
  [11] .plt              PROGBITS         00000000000004d0  000004d0
       0000000000000010  0000000000000010  AX       0     0     16
  [12] .plt.got          PROGBITS         00000000000004e0  000004e0
       0000000000000008  0000000000000008  AX       0     0     8
  [13] .text             PROGBITS         00000000000004f0  000004f0
       0000000000000192  0000000000000000  AX       0     0     16
  [14] .fini             PROGBITS         0000000000000684  00000684
       0000000000000009  0000000000000000  AX       0     0     4
```

```
  [15] .rodata           PROGBITS         0000000000000690  00000690
       0000000000000004  0000000000000004  AM       0     0     4
  [16] .eh_frame_hdr     PROGBITS         0000000000000694  00000694
       000000000000003c  0000000000000000   A       0     0     4
  [17] .eh_frame         PROGBITS         00000000000006d0  000006d0
       0000000000000108  0000000000000000   A       0     0     8
  [18] .init_array       INIT_ARRAY       0000000000200e18  00000e18
       0000000000000008  0000000000000008  WA       0     0     8
  [19] .fini_array       FINI_ARRAY       0000000000200e20  00000e20
       0000000000000008  0000000000000008  WA       0     0     8
  [20] .dynamic          DYNAMIC          0000000000200e28  00000e28
       00000000000001b0  0000000000000010  WA       6     0     8
  [21] .got              PROGBITS         0000000000200fd8  00000fd8
       0000000000000028  0000000000000008  WA       0     0     8
  [22] .got.plt          PROGBITS         0000000000201000  00001000
       0000000000000018  0000000000000008  WA       0     0     8
  [23] .data             PROGBITS         0000000000201018  00001018
       0000000000000010  0000000000000000  WA       0     0     8
  [24] .bss              NOBITS           0000000000201028  00001028
       0000000000000008  0000000000000000  WA       0     0     1
  [25] .comment          PROGBITS         0000000000000000  00001028
       000000000000001c  0000000000000001  MS       0     0     1
  [26] .symtab           SYMTAB           0000000000000000  00001048
       00000000000005d0  0000000000000018          27    43     8
  [27] .strtab           STRTAB           0000000000000000  00001618
       00000000000001f5  0000000000000000          0     0     1
  [28] .shstrtab         STRTAB           0000000000000000  0000180d
       00000000000000fd  0000000000000000          0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  l (large), p (processor specific)
```

Листинг 3.5: Вывод содержимого Section Header Table

Полученные разделы содержат программу и различную информацию, например:

- .bss - в этом разделе содержаться не инициализированные данные, но под которые уже выделено место, заполненное нулями;
- .dynsym - в этом разделе содержится таблица символов динамического связывания, или «Symbol Table», содержит только глобальные символы;
- .symtab - раздел аналогичен .dynsym, но содержит уже все символы, а не только глобальные;
- .init - В этом разделе содержатся исполняемые инструкции, которые участвуют в инициализации программы. То есть, когда программа запускается, система организует выполнение кода в этом разделе перед вызовом основной точки входа в программу (называемой **main** для программ C).
- ...

Можно сказать что dynsym является уменьшенной версией symtab, которая содержит лишь глобальные символы. По информации написанной в блоге **Oracle**[3], возможно наличие и одной таблицы символов, но в таком случае потребуется больше памяти.

Рассмотрим что хранится в разделах **.dynsym** и **.symtab**.

```
root@kali:~/Desktop/testFolder# readelf -s example.out

Symbol table '.dynsym' contains 6 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND
    ↪ _ITM_deregisterTMCloneTab
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    ↪ __libc_start_main@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     4: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND
    ↪ _ITM_registerTMCloneTable
     5: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND
    ↪ __cxa_finalize@GLIBC_2.2.5 (2)

Symbol table '.symtab' contains 62 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000238     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000000254     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000000274     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000298     0 SECTION LOCAL  DEFAULT    4
     5: 00000000000002b8     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000348     0 SECTION LOCAL  DEFAULT    6
     7: 00000000000003c6     0 SECTION LOCAL  DEFAULT    7
     8: 00000000000003d8     0 SECTION LOCAL  DEFAULT    8
     9: 00000000000003f8     0 SECTION LOCAL  DEFAULT    9
    10: 00000000000004b8     0 SECTION LOCAL  DEFAULT   10
    11: 00000000000004d0     0 SECTION LOCAL  DEFAULT   11
    12: 00000000000004e0     0 SECTION LOCAL  DEFAULT   12
    13: 00000000000004f0     0 SECTION LOCAL  DEFAULT   13
    14: 0000000000000684     0 SECTION LOCAL  DEFAULT   14
    15: 0000000000000690     0 SECTION LOCAL  DEFAULT   15
    16: 0000000000000694     0 SECTION LOCAL  DEFAULT   16
    17: 00000000000006d0     0 SECTION LOCAL  DEFAULT   17
    18: 0000000000200e18     0 SECTION LOCAL  DEFAULT   18
    19: 0000000000200e20     0 SECTION LOCAL  DEFAULT   19
    20: 0000000000200e28     0 SECTION LOCAL  DEFAULT   20
    21: 0000000000200fd8     0 SECTION LOCAL  DEFAULT   21
    22: 0000000000201000     0 SECTION LOCAL  DEFAULT   22
    23: 0000000000201018     0 SECTION LOCAL  DEFAULT   23
    24: 0000000000201028     0 SECTION LOCAL  DEFAULT   24
    25: 0000000000000000     0 SECTION LOCAL  DEFAULT   25
    26: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    27: 0000000000000520     0 FUNC    LOCAL  DEFAULT   13 deregister_tm_clones
    28: 0000000000000560     0 FUNC    LOCAL  DEFAULT   13 register_tm_clones
    29: 00000000000005b0     0 FUNC    LOCAL  DEFAULT   13
    ↪ __do_global_dtors_aux
    30: 0000000000201028     1 OBJECT  LOCAL  DEFAULT   24 completed.7001
    31: 0000000000200e20     0 OBJECT  LOCAL  DEFAULT   19
    ↪ __do_global_dtors_aux_fin
    32: 00000000000005f0     0 FUNC    LOCAL  DEFAULT   13 frame_dummy
    33: 0000000000200e18     0 OBJECT  LOCAL  DEFAULT   18
    ↪ __frame_dummy_init_array_
    34: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS example.cpp
    35: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    36: 00000000000007d4     0 OBJECT  LOCAL  DEFAULT   17 __FRAME_END__
    37: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS
    38: 0000000000200e20     0 NOTYPE  LOCAL  DEFAULT   18 __init_array_end
    39: 0000000000200e28     0 OBJECT  LOCAL  DEFAULT   20 _DYNAMIC
```

```
54     40: 0000000000200e18     0 NOTYPE  LOCAL  DEFAULT    18 __init_array_start
55     41: 0000000000000694     0 NOTYPE  LOCAL  DEFAULT    16 __GNU_EH_FRAME_HDR
56     42: 0000000000201000     0 OBJECT  LOCAL  DEFAULT    22
       ↪ _GLOBAL_OFFSET_TABLE_
57     43: 0000000000000680     2 FUNC    GLOBAL DEFAULT    13 __libc_csu_fini
58     44: 0000000000000000     0 NOTYPE  WEAK   DEFAULT   UND
       ↪ _ITM_deregisterTMCloneTab
59     45: 0000000000201018     0 NOTYPE  WEAK   DEFAULT    23 data_start
60     46: 0000000000201028     0 NOTYPE  GLOBAL DEFAULT    23 _edata
61     47: 0000000000000684     0 FUNC    GLOBAL DEFAULT    14 _fini
62     48: 0000000000000000     0 FUNC    GLOBAL DEFAULT   UND
       ↪ __libc_start_main@@GLIBC_
63     49: 0000000000201018     0 NOTYPE  GLOBAL DEFAULT    23 __data_start
64     50: 0000000000000000     0 NOTYPE  WEAK   DEFAULT   UND __gmon_start__
65     51: 0000000000201020     0 OBJECT  GLOBAL HIDDEN     23 __dso_handle
66     52: 0000000000000690     4 OBJECT  GLOBAL DEFAULT    15 _IO_stdin_used
67     53: 0000000000000610   101 FUNC    GLOBAL DEFAULT    13 __libc_csu_init
68     54: 0000000000201030     0 NOTYPE  GLOBAL DEFAULT    24 _end
69     55: 00000000000004f0    43 FUNC    GLOBAL DEFAULT    13 _start
70     56: 0000000000201028     0 NOTYPE  GLOBAL DEFAULT    24 __bss_start
71     57: 00000000000005fa    11 FUNC    GLOBAL DEFAULT    13 main
72     58: 0000000000201028     0 OBJECT  GLOBAL HIDDEN     23 __TMC_END__
73     59: 0000000000000000     0 NOTYPE  WEAK   DEFAULT   UND
       ↪ _ITM_registerTMCloneTable
74     60: 0000000000000000     0 FUNC    WEAK   DEFAULT   UND
       ↪ __cxa_finalize@@GLIBC_2.2
75     61: 00000000000004b8     0 FUNC    GLOBAL DEFAULT    10 _init
```

Листинг 3.6: Вывод содержимого раздела .dynsym

Как видно из листинга выше, были выведены все символы из объектного файла. Стоит заметить что присутствуют все символы из вывода листинга 3.2. Это дает понять, откуда утилита **nm** берет таблицу символов, еще до разбора её исходного кода.

3.2 Адрес символа

Виртуальный адрес символа, выводится в первом столбце вывода утилиты nm. В некоторых случаях, этот адрес может быть не определен.

Если проанализировать адресное пространство символов из листинга 3.2, то можно заметить, что весь список адресов символов находится между адресами символов _start и _end.

3.3 Типы символов

Если символ написан маленькими буквами, то он локальный. Иначе он глобальный (внешний).

Символ	Описание
A	Значение символа является абсолютным и не будет изменено дальнейшей линковкой.
B b	Символ находится в разделе не инициализированных данных(BSS раздел).

C	Символ является общим. Общие символы - не инициализированные данные. При компоновке могут отображаться несколько общих символов с тем же именем.
D d	Символ находится в секции инициализированных данных.
G g	Символ находится в инициализированной секции данных для небольших объектов. Некоторые форматы объектных файлов обеспечивают более эффективный доступ к небольшим объектам данных, таким как глобальная переменная int, а не к большому глобальному массиву.
i	Для файлов формата PE это означает, что символ находится в разделе, посвященном реализации DLL. Для файлов формата ELF это означает, что символ является косвенной функцией. Это расширение GNU для стандартного набора типов символов ELF.
N	Символ является отладочным символом.
p	Символы находятся в секции размотки стека.
R r	Символ находится в разделе данных только для чтения.
S s	Символ находится в не инициализированной секции данных для небольших объектов.
T t	Символ находится в разделе **text**.
U	Символ не определен.
u	Символ является уникальным глобальным символом. Это расширение GNU для стандартного набора привязок символов ELF. Для такого символа динамический компоновщик гарантирует, что во всем процессе есть только один символ с этим именем и типом в использовании.
V v	Символ - слабый объект. Когда слабый определенный символ связан с нормальным определенным символом, нормальный определенный символ используется без ошибок. Когда слабый неопределенный символ связан и символ не определен, значение слабого символа становится равным нулю без ошибок. В некоторых системах верхний регистр указывает, что указано значение по умолчанию.
W w	Символ является слабым символом, который не был специально помечен как символ слабого объекта. Когда слабый определенный символ связан с нормальным определенным символом, нормальный определенный символ используется без ошибок. Когда слабый неопределенный символ связан и символ не определен, значение символа определяется системным образом без ошибок. В некоторых системах верхний регистр указывает, что указано значение по умолчанию.
-	Отладочная информация.
?	Тип символа неизвестен.

3.4 Имя символов

Имеются несколько типов имени:

- пользовательского уровня;
- пользовательского уровня с дополнениями компилятора;

- добавленные компилятором для корректной работы программы.

3.5 Аргументы утилиты

Ключ	Описание
-A -o	Выделяет каждый символ имени входного файла в котором он был найден, до идентификации входного файла только, перед всеми его символами.
-a	Отображает все отладочные символы, по умолчанию они скрыты.
-B	Выходной формат bsd.
-C	Преобразует имена низкоуровневых символов в имена пользовательского уровня.
-D	Отображает все динамические символы вместо обычных символов.
-f format	Использует выходной формат который может быть bsd, sysv или posix. Только первый символ **format**-а имеет значение.
-g	Отображает только внешние символы.
-n -v	Сортирует символы по их адресам, до алфавитного упорядочивания.
-p	Отменяет какой либо порядок сортировки. Печатает символы в порядке поступления.
-S	Вывод размера определенных символов.
-r	Меняет порядок сортировки на обратный (как числовой так и алфавитный).
-t radix	Использовать RADIX как основание системы счисления для печати значения символов. Десятичной системе соответствует ’d’, восьмеричной - ’o’, шестнадцатеричной - ’x’.
-u	Отображает все отладочные символы, обычные символы не отображаются.
-V	Показать номер версии nm и завершить работу.
–help	Показать список опций nm и завершить работу.

3.6 Примеры работы

Для большего вывода утилиты **nm** была написана следующая программа.

```
1  #include <stdio.h>
2
3  const int myConstInt=5;
4  int a;
5  int b = 10;
6  int *c;
7  int *d = NULL;
8
9  extern short e;
10 extern short *f;
11
12 char g[5];
13 char h[6] = "Denis";
```

```
14 extern char i[2];
15
16 static int p;
17 static int q = 40;
18 static int r[4];
19 static int s[4] = {1, 2, 3};
20
21 int main()
22 {
23         int j;
24         int k = 30;
25         int *l;
26         int *m = NULL;
27         int n[3];
28         int o[3] = {4, 0};
29 }
```

Листинг 3.7: example2.cpp

Далее данная программа была скомпилирована в объектный файл(example2.out).

3.6.1 Запуск без дополнительных ключей

Утилита nm была запущена для вывода таблицы символов файла example2.o.

```
1  root@kali:~/Desktop/testFolder# g++ example2.cpp -o example2.o
2  root@kali:~/Desktop/testFolder# nm example2.o
3  0000000000200df8 d _DYNAMIC
4  0000000000201000 d _GLOBAL_OFFSET_TABLE_
5  00000000000006d0 R _IO_stdin_used
6                   w _ITM_deregisterTMCloneTable
7                   w _ITM_registerTMCloneTable
8  00000000000006d4 r _ZL10myConstInt
9  0000000000201080 b _ZL1p
10 000000000020103c d _ZL1q
11 0000000000201090 b _ZL1r
12 0000000000201040 d _ZL1s
13 000000000000081c r __FRAME_END__
14 00000000000006d8 r __GNU_EH_FRAME_HDR
15 0000000000201050 D __TMC_END__
16 0000000000201050 B __bss_start
17                  w __cxa_finalize@@GLIBC_2.2.5
18 0000000000201020 D __data_start
19 00000000000005d0 t __do_global_dtors_aux
20 0000000000200df0 t __do_global_dtors_aux_fini_array_entry
21 0000000000201028 D __dso_handle
22 0000000000200de8 t __frame_dummy_init_array_entry
23                  w __gmon_start__
24 0000000000200df0 t __init_array_end
25 0000000000200de8 t __init_array_start
26 00000000000006c0 T __libc_csu_fini
27 0000000000000650 T __libc_csu_init
28                  U __libc_start_main@@GLIBC_2.2.5
29 0000000000201050 D _edata
30 00000000002010a0 B _end
31 00000000000006c4 T _fini
32 00000000000004d8 T _init
33 0000000000000510 T _start
34 0000000000201060 B a
35 0000000000201030 D b
```

```
0000000000201068 B c
0000000000201050 b completed.7001
0000000000201070 B d
0000000000201020 W data_start
0000000000000540 t deregister_tm_clones
0000000000000610 t frame_dummy
0000000000201078 B g
0000000000201034 D h
000000000000061a T main
0000000000000580 t register_tm_clones
```

Листинг 3.8: Вывод утилиты nm

Сперва разберем какие типы символов получили переменные в программе.

Переменная	Тип символа(nm)	Комментарий
int b=10;	D	Инициализированные переменные попали в секцию инициализированных данных.
char h[6] = ”Denis”;		
static int q = 40;		
static int s[4] = 1, 2, 3;		
int a;	B	Не инициализированные переменные попали в секцию не инициализированных данных.
int *c;		
int *d;		
char g[5];		
static int p;		
static int r[4];		
const int myConstInt=5;	r	Символ находится в разделе данных доступных только для чтения.

Переменные типа extern не были добавлены в таблицу символов, так как ключевое слово **extern** подразумевает что переменные объявляются где-то в другом месте(другом файле с кодом), но такой файл с кодом не был предоставлен и соответственно в таблице символов нет информации об этих переменных.

Для некоторых переменных были изменены названия, это связано с особенностями работы компилятора, в частности для решения проблем уникальности имен.

Так-же в таблице символов, множество сторонних символов, рассмотрим что из себя представляют некоторые из них:

- **_GLOBAL_OFFSET_TABLE_** - используется для нахождения реальных адресов глобальных переменных;
- **__GNU_EH_FRAME_HDR** - используется c++ для доступа к eh_frame, который в свою очередь содержит информацию об исключениях;
- **__bss_start** - раздел содержащий не инициализированные данные, но под которые уже выделено место, заполненное нулями;
- **__do_global_dtors_aux** - функция вызывающая деструкторы объектов.

Таким образом, в таблице символов, помимо пользовательских присутствуют еще и дополнительные, добавленные компилятором для корректной работы программы.

3.6.2 Ключ -a

Отображает все отладочные символы, по умолчанию они скрыты.

```
root@kali:~/Desktop/testFolder# nm -a example2.o
0000000000000000 a
0000000000201050 b .bss
0000000000000000 n .comment
0000000000201020 d .data
0000000000200df8 d .dynamic
0000000000000348 r .dynstr
00000000000002b8 r .dynsym
0000000000000718 r .eh_frame
00000000000006d8 r .eh_frame_hdr
00000000000006c4 t .fini
0000000000200df0 t .fini_array
0000000000000298 r .gnu.hash
00000000000003ec r .gnu.version
00000000000003f8 r .gnu.version_r
0000000000200fd8 d .got
0000000000201000 d .got.plt
00000000000004d8 t .init
0000000000200de8 t .init_array
0000000000000238 r .interp
0000000000000254 r .note.ABI-tag
0000000000000274 r .note.gnu.build-id
00000000000004f0 t .plt
0000000000000500 t .plt.got
0000000000000418 r .rela.dyn
00000000000006d0 r .rodata
0000000000000510 t .text
0000000000200df8 d _DYNAMIC
0000000000201000 d _GLOBAL_OFFSET_TABLE_
00000000000006d0 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000006d4 r _ZL10myConstInt
0000000000201080 b _ZL1p
000000000020103c d _ZL1q
0000000000201090 b _ZL1r
0000000000201040 d _ZL1s
000000000000081c r __FRAME_END__
00000000000006d8 r __GNU_EH_FRAME_HDR
0000000000201050 D __TMC_END__
0000000000201050 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000201020 D __data_start
00000000000005d0 t __do_global_dtors_aux
0000000000200df0 t __do_global_dtors_aux_fini_array_entry
0000000000201028 D __dso_handle
0000000000200de8 t __frame_dummy_init_array_entry
                 w __gmon_start__
0000000000200df0 t __init_array_end
0000000000200de8 t __init_array_start
00000000000006c0 T __libc_csu_fini
0000000000000650 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000201050 D _edata
00000000002010a0 B _end
00000000000006c4 T _fini
00000000000004d8 T _init
```

```
0000000000000510 T _start
0000000000201060 B a
0000000000201030 D b
0000000000201068 B c
0000000000201050 b completed.7001
0000000000000000 a crtstuff.c
0000000000000000 a crtstuff.c
0000000000201070 B d
0000000000201020 W data_start
0000000000000540 t deregister_tm_clones
0000000000000000 a example2.cpp
0000000000000610 t frame_dummy
0000000000201078 B g
0000000000201034 D h
000000000000061a T main
0000000000000580 t register_tm_clones
```

Листинг 3.9: Вывод утилиты nm с ключом -a

В список символов были добавлены разделы с отладочной информацией, которые также присутствуют в Section Header Table ELF файла.

3.6.3 Ключ -C

Преобразует имена низкоуровневых символов в имена пользовательского уровня.

```
root@kali:~/Desktop/testFolder# nm –C example2.o
0000000000200df8 d _DYNAMIC
0000000000201000 d _GLOBAL_OFFSET_TABLE_
00000000000006d0 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000006d4 r myConstInt
0000000000201080 b p
000000000020103c d q
0000000000201090 b r
0000000000201040 d s
000000000000081c r __FRAME_END__
00000000000006d8 r __GNU_EH_FRAME_HDR
0000000000201050 D __TMC_END__
0000000000201050 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000201020 D __data_start
00000000000005d0 t __do_global_dtors_aux
0000000000200df0 t __do_global_dtors_aux_fini_array_entry
0000000000201028 D __dso_handle
0000000000200de8 t __frame_dummy_init_array_entry
                 w __gmon_start__
0000000000200df0 t __init_array_end
0000000000200de8 t __init_array_start
00000000000006c0 T __libc_csu_fini
0000000000000650 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000201050 D _edata
00000000002010a0 B _end
00000000000006c4 T _fini
00000000000004d8 T _init
0000000000000510 T _start
0000000000201060 B a
0000000000201030 D b
```

```
0000000000201068 B c
0000000000201050 b completed.7001
0000000000201070 B d
0000000000201020 W data_start
0000000000000540 t deregister_tm_clones
0000000000000610 t frame_dummy
0000000000201078 B g
0000000000201034 D h
000000000000061a T main
0000000000000580 t register_tm_clones
```

Листинг 3.10: Вывод утилиты nm с ключом -C

Действительно, имена некоторых символов изменились, например имя **_ZL1p** преобразовалось в **p**. Данный ключ успешно отделил название символа определенное пользователем от того что дописал компилятор.

3.6.4 Ключ -D

Отображение только динамических символов.

```
root@kali:~/Desktop/testFolder# nm –D example2.o
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
                 w __cxa_finalize
                 w __gmon_start__
                 U __libc_start_main
```

Листинг 3.11: Вывод утилиты nm с ключом -D

Как и ожидалось в выводе остались лишь динамические символы, хоть и неопределенные.

3.6.5 Ключ -f

По умолчанию вывод происходит в формате **bsd**, но также возможен в форматах **sysv** и **posix**.

```
root@kali:~/Desktop/testFolder# nm –f posix example2.o
_DYNAMIC d 0000000000200df8
_GLOBAL_OFFSET_TABLE_ d 0000000000201000
_IO_stdin_used R 00000000000006d0 0000000000000004
_ITM_deregisterTMCloneTable w
_ITM_registerTMCloneTable w
_ZL10myConstInt r 00000000000006d4 0000000000000004
_ZL1p b 0000000000201080 0000000000000004
_ZL1q d 000000000020103c 0000000000000004
_ZL1r b 0000000000201090 0000000000000010
_ZL1s d 0000000000201040 0000000000000010
__FRAME_END__ r 000000000000081c
__GNU_EH_FRAME_HDR r 00000000000006d8
__TMC_END__ D 0000000000201050
__bss_start B 0000000000201050
__cxa_finalize@@GLIBC_2.2.5 w
__data_start D 0000000000201020
__do_global_dtors_aux t 00000000000005d0
__do_global_dtors_aux_fini_array_entry t 0000000000200df0
__dso_handle D 0000000000201028
__frame_dummy_init_array_entry t 0000000000200de8
```

```
__gmon_start__ w
__init_array_end t 0000000000200df0
__init_array_start t 0000000000200de8
__libc_csu_fini T 00000000000006c0 0000000000000002
__libc_csu_init T 0000000000000650 0000000000000065
__libc_start_main@@GLIBC_2.2.5 U
_edata D 0000000000201050
_end B 00000000002010a0
_fini T 00000000000006c4
_init T 00000000000004d8
_start T 0000000000000510 000000000000002b
a B 0000000000201060 0000000000000004
b D 0000000000201030 0000000000000004
c B 0000000000201068 0000000000000008
completed.7001 b 0000000000201050 0000000000000001
d B 0000000000201070 0000000000000008
data_start W 0000000000201020
deregister_tm_clones t 0000000000000540
frame_dummy t 0000000000000610
g B 0000000000201078 0000000000000005
h D 0000000000201034 0000000000000006
main T 000000000000061a 0000000000000030
register_tm_clones t 0000000000000580
```

Листинг 3.12: Вывод утилиты nm с ключом -f posix

Вывод изменился, теперь спервa идет имя символа, далее его тип, затем адрес и новое поле - размер.

```
root@kali:~/Desktop/testFolder# nm -f sysv example2.o


Symbols from example2.o:

Name                  Value           Class        Type         Size
    ↪  Line  Section

_DYNAMIC              |0000000000200df8|   d  |              OBJECT|
    ↪  |      |.dynamic
_GLOBAL_OFFSET_TABLE_|0000000000201000|   d  |               OBJECT|
    ↪  |      |.got.plt
_IO_stdin_used        |00000000000006d0|   R  |              OBJECT
    ↪ |0000000000000004|      |.rodata
_ITM_deregisterTMCloneTable|                  |   w  |              NOTYPE|
    ↪          |      |*UND*
_ITM_registerTMCloneTable|                    |   w  |              NOTYPE|
    ↪          |      |*UND*
_ZL10myConstInt       |00000000000006d4|   r  |              OBJECT
    ↪ |0000000000000004|      |.rodata
_ZL1p                 |0000000000201080|   b  |              OBJECT
    ↪ |0000000000000004|      |.bss
_ZL1q                 |000000000020103c|   d  |              OBJECT
    ↪ |0000000000000004|      |.data
_ZL1r                 |0000000000201090|   b  |              OBJECT
    ↪ |0000000000000010|      |.bss
_ZL1s                 |0000000000201040|   d  |              OBJECT
    ↪ |0000000000000010|      |.data
__FRAME_END__         |000000000000081c|   r  |              OBJECT|
    ↪  |      |.eh_frame
__GNU_EH_FRAME_HDR   |00000000000006d8|   r  |              NOTYPE|
    ↪  |      |.eh_frame_hdr
```

```
20 |__TMC_END__          |0000000000201050|   D  |            OBJECT|
   ↪  |      |.data
21 |__bss_start          |0000000000201050|   B  |            NOTYPE|
   ↪  |      |.bss
22 |__cxa_finalize@@GLIBC_2.2.5|                |   w  |              FUNC|
   ↪          |      |*UND*
23 |__data_start         |0000000000201020|   D  |            NOTYPE|
   ↪  |      |.data
24 |__do_global_dtors_aux|00000000000005d0|   t  |              FUNC|
   ↪    |      |.text
25 |__do_global_dtors_aux_fini_array_entry|0000000000200df0|   t  |
   ↪ OBJECT|                |      |.fini_array
26 |__dso_handle         |0000000000201028|   D  |            OBJECT|
   ↪  |      |.data
27 |__frame_dummy_init_array_entry|0000000000200de8|   t  |            OBJECT|
   ↪             |      |.init_array
28 |__gmon_start__       |                |   w  |            NOTYPE|
   ↪  |      |*UND*
29 |__init_array_end     |0000000000200df0|   t  |            NOTYPE|
   ↪  |      |.init_array
30 |__init_array_start   |0000000000200de8|   t  |            NOTYPE|
   ↪  |      |.init_array
31 |__libc_csu_fini      |00000000000006c0|   T  |              FUNC
   ↪ |0000000000000002|      |.text
32 |__libc_csu_init      |0000000000000650|   T  |              FUNC
   ↪ |0000000000000065|      |.text
33 |__libc_start_main@@GLIBC_2.2.5|                |   U  |              FUNC|
   ↪             |      |*UND*
34 |_edata               |0000000000201050|   D  |            NOTYPE|
   ↪  |      |.data
35 |_end                 |00000000002010a0|   B  |            NOTYPE|
   ↪  |      |.bss
36 |_fini                |00000000000006c4|   T  |              FUNC|
   ↪  |      |.fini
37 |_init                |00000000000004d8|   T  |              FUNC|
   ↪  |      |.init
38 |_start               |0000000000000510|   T  |              FUNC|000000000000002
   ↪ b|      |.text
39 |a                    |0000000000201060|   B  |            OBJECT
   ↪ |0000000000000004|      |.bss
40 |b                    |0000000000201030|   D  |            OBJECT
   ↪ |0000000000000004|      |.data
41 |c                    |0000000000201068|   B  |            OBJECT
   ↪ |0000000000000008|      |.bss
42 |completed.7001       |0000000000201050|   b  |            OBJECT
   ↪ |0000000000000001|      |.bss
43 |d                    |0000000000201070|   B  |            OBJECT
   ↪ |0000000000000008|      |.bss
44 |data_start           |0000000000201020|   W  |            NOTYPE|
   ↪  |      |.data
45 |deregister_tm_clones|0000000000000540|   t  |              FUNC|
   ↪  |      |.text
46 |frame_dummy          |0000000000000610|   t  |              FUNC|
   ↪  |      |.text
47 |g                    |0000000000201078|   B  |            OBJECT
   ↪ |0000000000000005|      |.bss
48 |h                    |0000000000201034|   D  |            OBJECT
   ↪ |0000000000000006|      |.data
49 |main                 |000000000000061a|   T  |              FUNC
   ↪ |0000000000000030|      |.text
```

```
50 register_tm_clones   |0000000000000580|    t    |                FUNC|
     ↪   |       |.text
```

Листинг 3.13: Вывод утилиты nm с ключом -f sysv

Вывод стал заметно больше, добавились столбцы: тип, размер, линия, секция. Инициализированные символы(переменные из программы) оказались в секции **.data**, а не инициализированные в секции **.bss**.

Примечание: необязательно писать полностью имя формата, достаточно лишь первой буквы.

3.6.6 Ключ -n

Сортировка символов по их адресу.

```
1  root@kali:~/Desktop/testFolder# nm –n example2.o
2                   w _ITM_deregisterTMCloneTable
3                   w _ITM_registerTMCloneTable
4                   w __cxa_finalize@@GLIBC_2.2.5
5                   w __gmon_start__
6                   U __libc_start_main@@GLIBC_2.2.5
7  00000000000004d8 T _init
8  0000000000000510 T _start
9  0000000000000540 t deregister_tm_clones
10 0000000000000580 t register_tm_clones
11 00000000000005d0 t __do_global_dtors_aux
12 0000000000000610 t frame_dummy
13 000000000000061a T main
14 0000000000000650 T __libc_csu_init
15 00000000000006c0 T __libc_csu_fini
16 00000000000006c4 T _fini
17 00000000000006d0 R _IO_stdin_used
18 00000000000006d4 r _ZL10myConstInt
19 00000000000006d8 r __GNU_EH_FRAME_HDR
20 000000000000081c r __FRAME_END__
21 0000000000200de8 t __frame_dummy_init_array_entry
22 0000000000200de8 t __init_array_start
23 0000000000200df0 t __do_global_dtors_aux_fini_array_entry
24 0000000000200df0 t __init_array_end
25 0000000000200df8 d _DYNAMIC
26 0000000000201000 d _GLOBAL_OFFSET_TABLE_
27 0000000000201020 D __data_start
28 0000000000201020 W data_start
29 0000000000201028 D __dso_handle
30 0000000000201030 D b
31 0000000000201034 D h
32 000000000020103c d _ZL1q
33 0000000000201040 d _ZL1s
34 0000000000201050 D __TMC_END__
35 0000000000201050 B __bss_start
36 0000000000201050 D _edata
37 0000000000201050 b completed.7001
38 0000000000201060 B a
39 0000000000201068 B c
40 0000000000201070 B d
41 0000000000201078 B g
42 0000000000201080 b _ZL1p
43 0000000000201090 b _ZL1r
44 00000000002010a0 B _end
```

Листинг 3.14: Вывод утилиты nm с ключом -n

Первыми оказались не инициализированные символы, далее символы в порядке возрастания адреса. Инициализированных символов до символа **_init** и после символа **_end** не оказалось.

3.6.7 Ключ -S

Вывод размера определенных символов.

```
root@kali:~/Desktop/testFolder# nm --S example2.o
0000000000200df8 d _DYNAMIC
0000000000201000 d _GLOBAL_OFFSET_TABLE_
00000000000006d0 0000000000000004 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000006d4 0000000000000004 r _ZL10myConstInt
0000000000201080 0000000000000004 b _ZL1p
000000000020103c 0000000000000004 d _ZL1q
0000000000201090 0000000000000010 b _ZL1r
0000000000201040 0000000000000010 d _ZL1s
000000000000081c r __FRAME_END__
00000000000006d8 r __GNU_EH_FRAME_HDR
0000000000201050 D __TMC_END__
0000000000201050 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000201020 D __data_start
00000000000005d0 t __do_global_dtors_aux
0000000000200df0 t __do_global_dtors_aux_fini_array_entry
0000000000201028 D __dso_handle
0000000000200de8 t __frame_dummy_init_array_entry
                 w __gmon_start__
0000000000200df0 t __init_array_end
0000000000200de8 t __init_array_start
00000000000006c0 0000000000000002 T __libc_csu_fini
0000000000000650 0000000000000065 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000201050 D _edata
00000000002010a0 B _end
00000000000006c4 T _fini
00000000000004d8 T _init
0000000000000510 000000000000002b T _start
0000000000201060 0000000000000004 B a
0000000000201030 0000000000000004 D b
0000000000201068 0000000000000008 B c
0000000000201050 0000000000000001 b completed.7001
0000000000201070 0000000000000008 B d
0000000000201020 W data_start
0000000000000540 t deregister_tm_clones
0000000000000610 t frame_dummy
0000000000201078 0000000000000005 B g
0000000000201034 0000000000000006 D h
000000000000061a 0000000000000030 T main
0000000000000580 t register_tm_clones
```

Листинг 3.15: Вывод утилиты nm с ключом -S

Переменным типа **int** было выделено 4 байта, переменной **char g[5];** было выделено 5 байт соответственно.

3.6.8 Ключ -t

Вывод адреса символов в восьмеричной, десятичной или шестнадцатеричной системе счисления.

```
root@kali:~/Desktop/testFolder# nm -t d example2.o
0000000002100728 d _DYNAMIC
0000000002101248 d _GLOBAL_OFFSET_TABLE_
0000000000001744 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
0000000000001748 r _ZL10myConstInt
0000000002101376 b _ZL1p
0000000002101308 d _ZL1q
0000000002101392 b _ZL1r
0000000002101312 d _ZL1s
0000000000002076 r __FRAME_END__
0000000000001752 r __GNU_EH_FRAME_HDR
0000000002101328 D __TMC_END__
0000000002101328 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
0000000002101280 D __data_start
0000000000001488 t __do_global_dtors_aux
0000000002100720 t __do_global_dtors_aux_fini_array_entry
0000000002101288 D __dso_handle
0000000002100712 t __frame_dummy_init_array_entry
                 w __gmon_start__
0000000002100720 t __init_array_end
0000000002100712 t __init_array_start
0000000000001728 T __libc_csu_fini
0000000000001616 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000002101328 D _edata
0000000002101408 B _end
0000000000001732 T _fini
0000000000001240 T _init
0000000000001296 T _start
0000000002101344 B a
0000000002101296 D b
0000000002101352 B c
0000000002101328 b completed.7001
0000000002101360 B d
0000000002101280 W data_start
0000000000001344 t deregister_tm_clones
0000000000001552 t frame_dummy
0000000002101368 B g
0000000002101300 D h
0000000000001562 T main
0000000000001408 t register_tm_clones
```

Листинг 3.16: Вывод утилиты nm с ключом -t

Был произведен вывод в десятичной системе счисления, значения адресов соответственно были переведены в эту систему счисления.

Обзор принципов работы утилиты

4.1 Трассировка вызовов

Для начала рассмотрим что задействует в своей работе данная утилита. Для этого воспользуемся утилитой **strace**, которая выполняет трассировку вызовов программы выполняя их два раза: один раз для трассировки, второй для самой программы.

Из-за обилия вывода трассировки, её лог приведён в файле **strace.log**, а далее показано какие вызовы программой были совершены.

```
root@kali:~/Desktop/testFolder# strace -c nm example2.o
0000000000200df8 d _DYNAMIC
0000000000201000 d _GLOBAL_OFFSET_TABLE_
00000000000006d0 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000006d4 r _ZL10myConstInt
0000000000201080 b _ZL1p
000000000020103c d _ZL1q
0000000000201090 b _ZL1r
0000000000201040 d _ZL1s
000000000000081c r __FRAME_END__
00000000000006d8 r __GNU_EH_FRAME_HDR
0000000000201050 D __TMC_END__
0000000000201050 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000201020 D __data_start
00000000000005d0 t __do_global_dtors_aux
0000000000200df0 t __do_global_dtors_aux_fini_array_entry
0000000000201028 D __dso_handle
0000000000200de8 t __frame_dummy_init_array_entry
                 w __gmon_start__
0000000000200df0 t __init_array_end
0000000000200de8 t __init_array_start
00000000000006c0 T __libc_csu_fini
0000000000000650 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000201050 D _edata
00000000002010a0 B _end
00000000000006c4 T _fini
00000000000004d8 T _init
0000000000000510 T _start
0000000000201060 B a
0000000000201030 D b
0000000000201068 B c
0000000000201050 b completed.7001
0000000000201070 B d
0000000000201020 W data_start
0000000000000540 t deregister_tm_clones
```

```
0000000000000610 t frame_dummy
0000000000201078 B g
0000000000201034 D h
000000000000061a T main
0000000000000580 t register_tm_clones
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 27.17    0.000966          22        43           write
  9.85    0.000350          23        15         1 open
  9.85    0.000350          18        19           mmap
  9.37    0.000333          14        23           read
  7.76    0.000276          17        16           fstat
  6.64    0.000236          20        12           mprotect
  5.23    0.000186          31         6           stat
  4.87    0.000173          11        16           lseek
  4.84    0.000172          12        14           close
  4.64    0.000165          83         2           getdents
  2.93    0.000104          10        10         9 access
  2.87    0.000102          26         4           lstat
  0.93    0.000033          17         2           fcntl
  0.84    0.000030          30         1           getrlimit
  0.79    0.000028          28         1           munmap
  0.68    0.000024           8         3           brk
  0.45    0.000016          16         1           readlink
  0.31    0.000011          11         1           arch_prctl
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.003555                   190        10 total
```

Листинг 4.1: Вывод утилиты nm с ключом -t

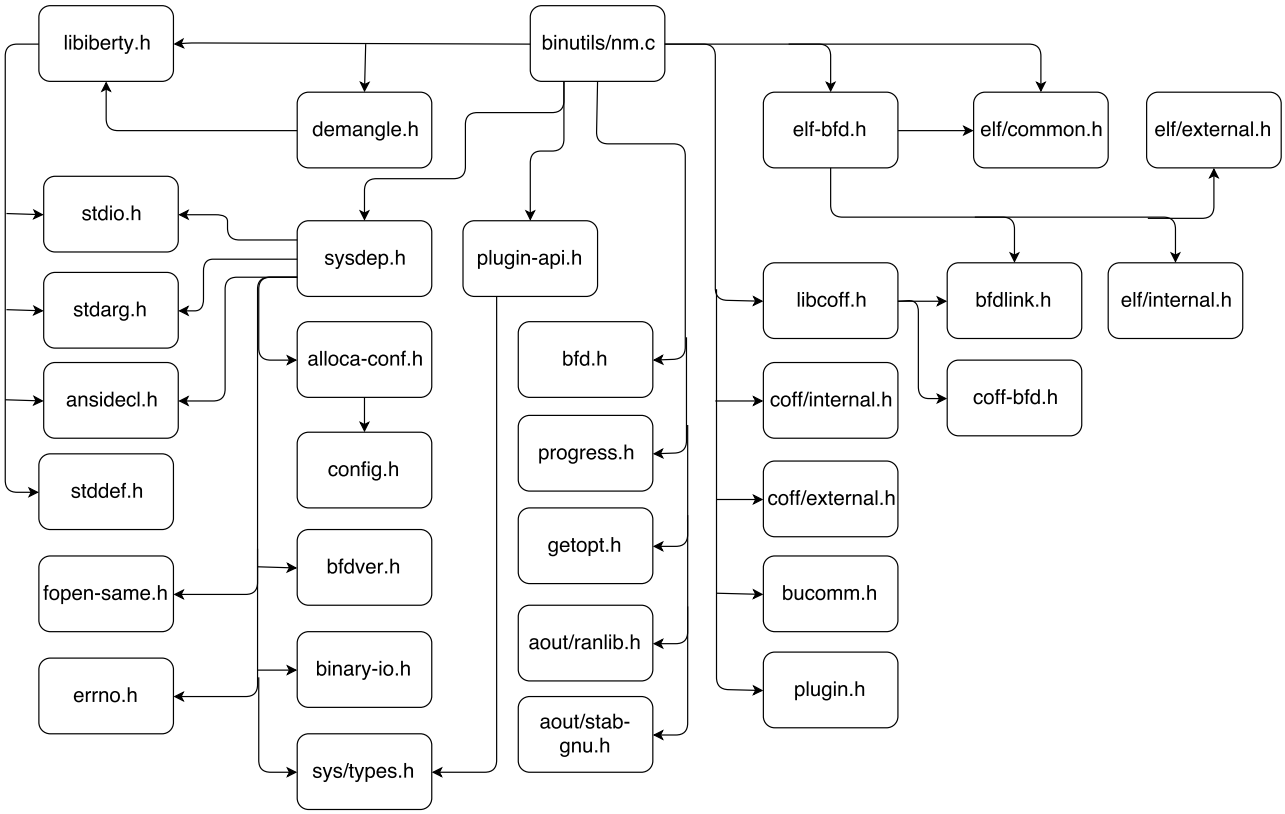
Как видно из результатов, 27.17% всего времени заняла печать результатов в консоль. Далее идут вызовы **open, mmap, read** - открытие, отображение в память и чтение файла соответственно.

Системный вызов	**Описание**
fstat	Получение информации о файле, по его дескриптору.
mprotect	Защита области памяти.
stat	Получение информации о файле, по его названию.
lseek	Получения данных по определенному смещению.
close	Закрытие потока работы с файлом.
getdents	Получение содержимого директории.
access	Проверка прав пользователя на доступ к файлу.
lstat	Идентично stat, но если вместо файла введена символическая ссылка(symbolic link), то будет получена инфомрация об этой ссылке.
fcntl	Манипуляции с файловым дескриптором.
getrlimit	Получение или задача лимитов ресурсов.
munmap	Удаление интервалов адресов из адресного пространства.
brk	Изменение размера сегмента данных.
readlink	Чтения значения символической ссылки.
arch_prctl	Установка архитектурно-специфичного состояния.
execve	Запуск программы.

4.2 Обзор исходного кода

Исходный код утилиты nm приложен к архиву, в папке source. Далее рассмотрено дерево зависимостей, заголовочные файлы, методы и алгоритм работы утилиты.

4.2.1 Дерево зависимостей



4.2.2 Подключаемые файлы заголовков

Рассмотрим какие заголовки подключены в коде утилиты.

Заголовок	Описание
libiberty.h	Объявление ряда функций, которые отсутствуют в некоторых операционных системах.
stdio.h	Стандартный заголовочный файл ввода-вывода.
stdarg.h	Средства для перебора аргументов функции, количество и типы которых заранее не известны.
ansideci.h	Набор макросов для работы с компилятором.
stddef.h	Определяет макросы NULL и offsetof, а также типы ptrdiff_t, wchar_t и size_t.
fopen-same.h	Макросы типов доступа к файлам, часть fopen, freopen or fdopen
errno.h	Макросы для идентификации ошибок через их код.
demangle.h	Отделение названия символа от символов, добавленных компилятором.
sysdep.h	Обработчик зависимостей пакета binutils.

alloca-conf.h	Автоматическая конфигурация кода используя скрипты.
config.h	Системный заголовок с конфигурацией системы.
bfdver.h	Информации о версии текущего пакета binutils.
binary-io.h	Бинарный ввод-вывод.
sys/types.h	Примитивные типы данных в системе.
plugin-api.h	Для работы компоновщика.
bfd.h	Часть пакета binutils. BFD расшифровывается как Binary File Descriptor library. Предоставляет удобный интерфейс для работы с частями объектного файла.
progress.h	Определения по умолчанию, для макроса прогресса.
getopt.h	Получение ключей запуска программы.
aout/stab_gnu.h	Содержит таблицу STAB кодов. Используется при отладке программ.
aout/ranlib.h	Определение смещения содержимого файла архива.
elf-bfd.h	Методы и структуры для работы с объектными(ELF) файлами.
elf/common.h	Идентификация переменных при работе с объектными файлами.
coff/external.h	Внешние COFF(Common Object File Format) структуры.
coff/internal.h	Внутренние COFF(Common Object File Format) структуры.
libcoff.h	Методы для работы с объектными(COFF) файлами.
bucomm.h	Методы для открытия файла.
plugin.h	Плагин-поддержка библиотеки BFD.
bfdlink.h.h	Работа с символами, определение типа, извлечение информации.
elf/internal.h	Отображение представление ELF файла(из оперативной памяти) в отображение библиотеки BFD.
elf/external.h	Отображение представление ELF файла(из файла) в отображение библиотеки BFD.

4.2.3 Методы утилиты

Рассмотрим какие методы содержит исходный код утилиты.

Имя метода	**Строка**	**Описание**
usage	237	help по утилите, выводит в консоль возможные ключи и их описание.
set_print_radix	291	Установка формата(системы счисления) для значения символа при выводе.
set_output_format	321	Установка общего формата вывода(BSD, SYSV, POSIX), по умолчанию используется формат BSD.
get_elf_symbol_type	346	Для ELF(Executable and Linking Format) типов файлов. Определение типа символа при выводе.

get_coff_symbol_type	375	Для COFF(Common Object File Format) типов файлов. Определение типа символа при выводе.
print_symname	408	Печать имени символа.
print_symdef_entry	426	Вывод индекса архива, а также обнаруженных символов.
filter_symbols	458	Фильтрация символов(внешних, локальных, определенных и т.д.) для вывода.
non_numeric_forward	548	Сравнение символов по их имени, для сравнения используется strcmp.
non_numeric_reverse	581	Инверсия прошлого метода.
numeric_forward	587	Сравнение двух значений символов. Используется для сортировки символов по их значению в порядке возрастания. Если значения символов одинаковы, то вызывается метод non_numeric_forward для сравнения их имен.
numeric_reverse	614	Инверсия прошлого метода.
size_forward1	637	Еще одно сравнение двух значений символов, в отличии от прошлых реализаций, иначе обрабатываются неопределенные символы
size_forward2	708	Определяет какой именно метод использовать для сортировки.
sort_symbols_by_size	727	Функция сортировки символов по их размеру.
get_relocs	825	Получение смещения для переданной секции.
print_symbol	859	Печать одного символа.
print_size_symbols	1021	Печать символов, при сортировки по размеру.
print_symbols	1060	Печать всех символов.
display_rel_file	1093	Печать символов объектного файла, с учетом введенных флагов.
set_print_width	1211	Задача длины строки(32 или 64 символа).
display_archive	1231	Печать символов файла архива.
display_file	1292	Определение типа файла и вызов метода display_rel_file или display_archive.
print_object_filename_bsd	1351	Печать имени файла. Для объектного файла, при выводе в формате bsd.
print_object_filename_sysv	1358	Печать имени файла, а также оглавления вывода. Для объектного файла, при выводе в формате bsd.
print_object_filename_posix	1373	Печать имени файла. Для объектного файла, при выводе в формате posix.

print_archive_filename_bsd	1382	Печать имени файла. Для файла архива, при выводе в формате bsd.
print_archive_filename_sysv	1389	Метод пуст.
print_archive_filename_posix	1394	Метод пуст.
print_archive_member_bsd	1401	Печать имени файла. Для файла-части архива, при выводе в формате bsd.
print_archive_member_sysv	1409	Печать имени файла. Для файла-части архива, при выводе в формате sysv.
print_archive_member_posix	1424	Печать имени файла. Для файла-части архива, при выводе в формате posix.
print_symbol_filename_bsd	1434	Печать имени файла(и архива если имеется), содержащего символ. Формат bsd.
print_symbol_filename_sysv	1445	Печать имени файла(и архива если имеется), содержащего символ. Формат sysv.
print_symbol_filename_posix	1456	Печать имени файла(и архива если имеется), содержащего символ. Формат posix.
print_value	1471	Печать значения символа.
print_symbol_info_bsd	1514	Печать одной строчки с информацией о символе, в формате bsd.
print_symbol_info_sysv	1554	Печать одной строчки с информацией о символе, в формате sysv.
print_symbol_info_posix	1609	Печать одной строчки с информацией о символе, в формате posix.
main	1626	Функция входа в программу. Анализирует введенные параметры, после чего вызывает функцию display_file или сообщает об ошибке.

4.2.4 Алгоритм работы

Рассмотрим как работает программа: от введения названия утилиты в консоли до выведенного результата.

1. Вход в функцию **main**;
   (a) Чтение введенных параметров, установка соответствующих флагов;
   (b) Проверка на несовместимые флаги;
   (c) Вызов функции **display_file**.
2. Функция **display_file**;
   (a) Проверка на существование файла, открытие его;
   (b) Определение типа файла(архив, объектный файл);
   (c) В зависимости от типа файла вызов функций **display_archive** или **display_rel_file**. Функция **display_archive** также использует функцию **display_rel_file** для каждого файла архива.
3. Функция **display_rel_file**;

(a) Проверка на наличие символов с помощью функции **bfd_read_minisymbols**. Также возвращается указатель на символы;

(b) Получение объема памяти, необходимого для хранения таблицы символов, используя функцию **bfd_get_dynamic_symtab_upper_bound** из **bfd.h**;

(c) Получение количества символов, используя функцию **bfd_canonicalize _dynamic_symtab** из **bfd.h**;

(d) Получения нового количества символов, после вызова функции фильтрации **filter_symbols**;

(e) Вызов функции **print_symbols** для печати символов.

Модификация программы

Модифицированный код утилиты nm приложен к архиву, в папке modification.

5.1 Модификация hello world!

Для начала произведем простейшую модификацию программы, с печатью в консоль фразы **hello world!**, при использовании нового ключа **-k**.

Для этого в строчке 1668, добавим в массив символов - символ k. Это позволит использовать данный ключ дальше.

```
1668   while ((c = getopt_long (argc, argv, "aABCDef:gHhlnkopPrSst:uvVvX:",
1669               long_options, (int *) 0)) != EOF)
1670     {
1671       switch (c)
1672     {
1673     case 'a':
```

Листинг 5.1: modification/nm.c

Теперь добавим соответствующую ветку(строка 1712) для switch. Для данного ключа, в консоль печатается сообщение **hello world!**, а дальше вызывается функция **exit(0)**, для завершения работы программы.

```
1712     case 'k':
1713       fprintf (stdout, "hello world!\n");
1714       exit (0);
```

Листинг 5.2: modification/nm.c

Перекомпилируем и запустим утилиту.

```
1 root@kali:~/opt/cross/x86_64-elf/bin# ./nm -k
2 hello world!
```

Листинг 5.3: Запуск модифицированной утилиты с ключом k

Чего и требовалось ожидать, в консоль вывелось сообщение **hello world!** и ничего более.

Рассмотрим подробнее как производилась перекомпиляция утилиты.

5.2 Перекомпиляция программы

Утилита nm является частью пакета binutils, соответственно в пакете находятся прочие утилиты и библиотеки для корректной работы. Все подключенные заголовочные файлы утилиты nm являются частью пакета binutils, поэтому перекомпилировать данную утилиты достаточно проблематично из-за большого количества зависимостей.

Поэтому для этих целей, в пакете присутствует файл **configure** в 15829 строк. Благодаря этому файлу имеется возможность сконфигурировать утилиты пакета для последующей сборки.

```
root@kali:~/Downloads/binutils-2.29# ./configure --target=x86_64-elf --prefix="
    ↪ $HOME/opt/cross" --disable-nls --disable-werror --disable-gdb --
    ↪ disable-libdecnumber --disable-readline --disable-sim
```

Листинг 5.4: Конфигурация утилит пакета

Конфигурация была запущена с различными ключами, рассмотрим их:

- **target** - указываем конечную архитектуру;
- **prefix** - директория, в которой будет конечный результат;
- **disable-nls** - отключение нативного языка системы, необходимо чтобы вывод ошибок производился на английском языке;
- **disable-werror** - перевод всех предупреждений в ошибки;
- оставшиеся ключи отключают ненужные возможности, для ускорения процесса перекомпиляции.

После конфигурации, можно запускать следующие команды:

- **make**;
- **make install**.

По окончанию перекомпиляции, исполняемые файлы программ будут находится в указанной в prefix папке, далее в папке соответствующей названию архитектуры, далее в папке bin.

То есть в данном случае, конечный путь будет следующим:

$HOME/opt/cross/x86_64-elf/bin/

5.3 Модификация ключа -C

Используя данный ключ, утилита преобразует модифицированные(demangled) имена символов в имена пользовательского уровня. Суть модификации состоит в том, чтобы помимо пользовательского имени вывести:

- модифицированное компилятором имя символа;
- имя символа пользовательского уровня;
- добавленные компилятором символы.

```
404 static char*
405 getDemangle(char *source, char *demangled){
406   char *ret = strstr(source, demangled);
407   int length = ret - source;
408   char *result = malloc(length);
409   int i=0;
410   for(;i<length;i++){
411     *(result+i)=*(source+i);
```

```
412     }
413   *(result+i)='\0';
414   return result;
415 }
416
417 /* Print symbol name NAME, read from ABFD, with printf format FORM,
418    demangling it if requested.  */
419
420 static void
421 print_symname (const char *form, const char *name, bfd *abfd)
422 {
423   if (do_demangle && *name)
424     {
425       char *res = bfd_demangle (abfd, name, DMGL_ANSI | DMGL_PARAMS);
426
427       if (res != NULL)
428     {
429       printf (" Demangled value: %s, source value: %s, demangle: %s",name, res,
    ↪  getDemangle(name, res));
430       free (res);
431       return;
432     }
433     }
434
435   printf (form, name);
436 }
```

Листинг 5.5: modification/nm.c

Метод отвечающий за печать символа находится в 421 строке, в его начале проверяется флаг **do_demangle**(преобразование к пользовательскому уровню), а также существование указателя на имя символа. Далее с помощью функции **do_demangle** указатель res получаеть уже символ пользовательского уровня.

Для того чтобы получить, какие символы добавил компилятор, был написан метод **getDemangle**, который находится в 405 строчке. Его алгоритм следующий:

1. с помощью функции **strstr** определяем указатель начало название символа пользовательского уровня;
2. получаем длины(количество) модифицированных символов;
3. выделяем память необходимой длины для этих символов;
4. в цикле заполняем выделенное место, необходимыми символами;
5. дописываем контрольный символ, для окончания записи;
6. возвращаем результат.

Далее с помощью **printf** происходит вывод:

- указателя **name** - модифицированное имя символа;
- указателя **res** - имя символа пользовательского уровня;
- результат функции **getDemangle** - добавленных компилятором символов.

Для примера была взята программа **example2.cpp** из листинга 3.7. Оригинальный вывод программы приведен в листинге 3.10.

```
root@kali:~/opt/cross/x86_64-elf/bin# ./nm example2.o -C
0000000000200df8 d _DYNAMIC
0000000000201000 d _GLOBAL_OFFSET_TABLE_
00000000000006d0 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000006d4 r Demangled value: _ZL10myConstInt, source value: myConstInt,
    ↪ demangle: _ZL10
0000000000201080 b Demangled value: _ZL1p, source value: p, demangle: _ZL1
000000000020103c d Demangled value: _ZL1q, source value: q, demangle: _ZL1
0000000000201090 b Demangled value: _ZL1r, source value: r, demangle: _ZL1
0000000000201040 d Demangled value: _ZL1s, source value: s, demangle: _ZL1
000000000000081c r __FRAME_END__
00000000000006d8 r __GNU_EH_FRAME_HDR
0000000000201050 D __TMC_END__
0000000000201050 B __bss_start
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000201020 D __data_start
00000000000005d0 t __do_global_dtors_aux
0000000000200df0 t __do_global_dtors_aux_fini_array_entry
0000000000201028 D __dso_handle
0000000000200de8 t __frame_dummy_init_array_entry
                 w __gmon_start__
0000000000200df0 t __init_array_end
0000000000200de8 t __init_array_start
00000000000006c0 T __libc_csu_fini
0000000000000650 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000201050 D _edata
00000000002010a0 B _end
00000000000006c4 T _fini
00000000000004d8 T _init
0000000000000510 T _start
0000000000201060 B a
0000000000201030 D b
0000000000201068 B c
0000000000201050 b completed.7001
0000000000201070 B d
0000000000201020 W data_start
0000000000000540 t deregister_tm_clones
0000000000000610 t frame_dummy
0000000000201078 B g
0000000000201034 D h
000000000000061a T main
0000000000000580 t register_tm_clones
```

Листинг 5.6: Вывод утилиты с модифицированным ключом -C

Чего и требовалось ожидать, у модифицированных имен символов появилась дополнительная информация.

5.4 Интеграция в систему

Ранее приведенные опыты вызывали скомпилированные утилиту, а не ту что стандартно используется в системе. Рассмотрим как заменить оригинальную утилиту на модифицированную на уровне системы.

```
root@kali:~# echo $PATH
```

```
2 /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Листинг 5.7: Вывод значения переменной PATH

Переменная окружения **PATH** позволяет вывести пути каталогов, в которых располагаются исполняемые файлы. Оригинальный исполняемый файл утилиты nm в моем случаем был расположен в каталоге **/usr/bin**.

С заменой в данной каталог был перемещен модифицированный, исполняемый файл утилиты. Затем была выполнена перезагрузка системы. После перезагрузки была выполнена проверка, на распознание ключа k.

```
1 root@kali:~# nm -k
2 hello world!
```

Листинг 5.8: Модифицированная утилита

Как и ожидалось оригинальная утилита была успешно заменена на модифицированную.

Вывод

В данной работе была изучена утилита nm. Были рассмотрены интерфейс утилиты, исходный код включая разбор написанных методов, алгоритм работы. Также была выполнена модификация утилиты, и замена оригинальной утилиты на собственную в используемой операционной системе.

Изучив трассировку вызов, было выяснено что большая часть времени тратиться на работу с файлами.

Из-за того что исходный код утилиты, сильно зависим от пакета binutils, отделение утилиты от пакета или серьёзная модификация исходного кода весьма затруднительны.

Столь простая модификация программ несколько пугает, например в стандартные утилиты пользователя можно встроить какие-нибудь перехватчики, просто заменив оригинальные программы на модифицированные. Да и сама тенденция открытого исходного кода, с его помощью можно посмотреть уязвимые места в программах.

Литература

[1] Утилиты обработки двочиных файлов [Электронный ресурс]. — URL: https://www.opennet.ru/docs/RUS/binutils/binutils-2.html (дата обращения: 2017-10-20).

[2] Executable and Linkable Format [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format (дата обращения: 2017-10-21).

[3] Inside ELF Symbol Tables [Электронный ресурс]. — URL: https://blogs.oracle.com/ali/inside-elf-symbol-tables (дата обращения: 2017-10-21).

[4] GNU Binary Utilities: nm [Электронный ресурс]. — URL: https://sourceware.org/binutils/docs/binutils/nm.html (дата обращения: 2017-10-21).

[5] Executable and Linkable Format (ELF) [Электронный ресурс]. — URL: http://www.skyfree.org/linux/references/ELF_Format.pdf (дата обращения: 2017-10-21).