使用readelf和objdump解析目标文件

1. 目标文件

目标文件的文件类型为ELF，在Linux下对应文件后缀为.o的文件，Windows下对应文件后缀为.obj的文件。使用file命令可以查看到.o和.obj文件均为ELF类型。

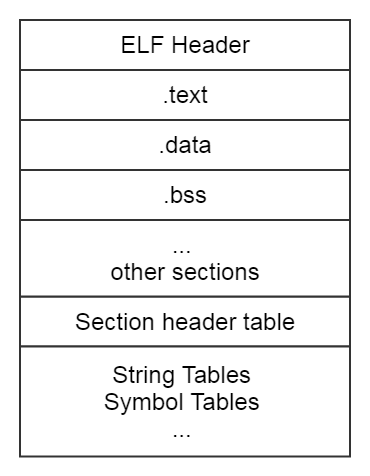
C:\Users\hitao_ma\AppData\Local\Temp\1521897691(1).png

目标文件只是ELF文件的可重定位文件(Relocatable file)，ELF文件一共有4种类型：Relocatable file、Executable file、Shared object file和Core Dump file{1什么是Core Dump：Core Dump 又叫核心转储。在程序运行过程中发生异常时，将其内存数据保存到文件中，这个过程叫做 Core Dump。2 Core Dump 的作用：在开发过程中，难免会遇到程序运行过程中异常退出的情况，这时候想要定位哪里出了问题，仅仅依靠程序自身的信息打印（日志记录）往往是不够的，这个时候就需要 Core Dump 文件来帮忙了。一个完整的 Core Dump 文件实际上相当于恢复了异常现场，利用 Core Dump 文件，可以查看到程序异常时的所有信息，变量值、栈信息、内存数据，程序异常时的运行位置（甚至记录代码行号）等等，定位所需要的一切信息都可以从 Core Dump文件获取到，能够非常有效的提高定位效率。}

我们可以使用工具readelf 和objdump对目标文件simple.o进行分析。为了加深对目标文件的理解，在使用readelf & objdump进行前，需要先要了解ELF文件的结构。

1. ELF文件结构

和class文件类似，ELF文件存放数据的格式也是固定的，计算机在解析目标文件时，就是按照它每个字段的数据结构进行逐字解析的。ELF文件结构信息定义在/usr/include/elf.h中，整个ELF文件的结构如下图：



* **ELF Header**

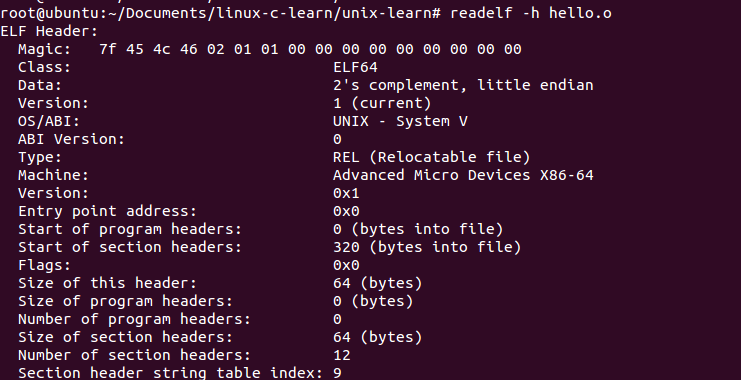
ELF Header是ELF文件的第一部分，64 bit的ELF文件头的结构体如下：

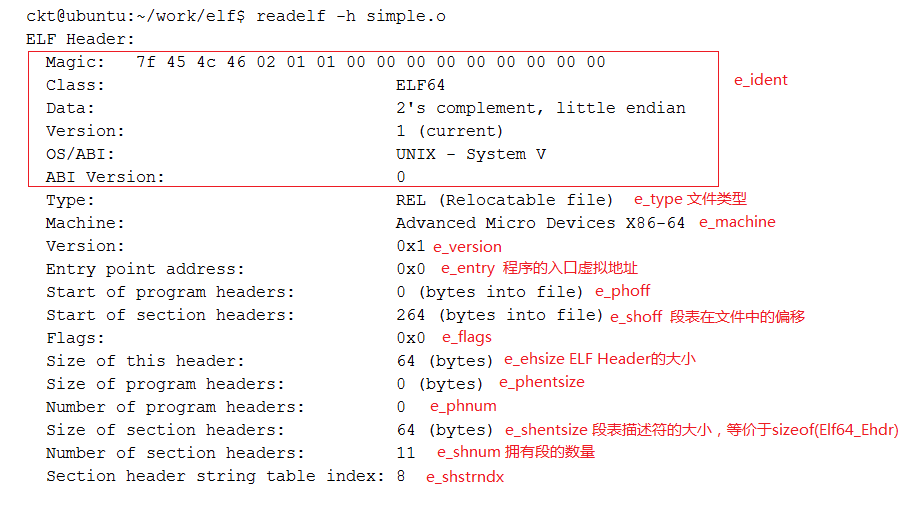
|  |
| --- |
| typedef struct  {  unsigned char e\_ident[EI\_NIDENT]; /\* Magic number and other info \*/  Elf64\_Half e\_type; /\* Object file type \*/  Elf64\_Half e\_machine; /\* Architecture \*/  Elf64\_Word e\_version; /\* Object file version \*/  Elf64\_Addr e\_entry; /\* Entry point virtual address \*/  Elf64\_Off e\_phoff; /\* Program header table file offset \*/  Elf64\_Off e\_shoff; /\* Section header table file offset \*/  Elf64\_Word e\_flags; /\* Processor-specific flags \*/  Elf64\_Half e\_ehsize; /\* ELF header size in bytes \*/  Elf64\_Half e\_phentsize; /\* Program header table entry size \*/  Elf64\_Half e\_phnum; /\* Program header table entry count \*/  Elf64\_Half e\_shentsize; /\* Section header table entry size \*/  Elf64\_Half e\_shnum; /\* Section header table entry count \*/  Elf64\_Half e\_shstrndx; /\* Section header string table index \*/  } Elf64\_Ehdr; |

接下来我们会使用到第一个分析目标文件的工具readelf，通过man

readelf命令，我们可以查到readelf的作用就是用来显示ELF文件的信息。

结合Elf64\_Ehdr来看，对应解析结果如下：



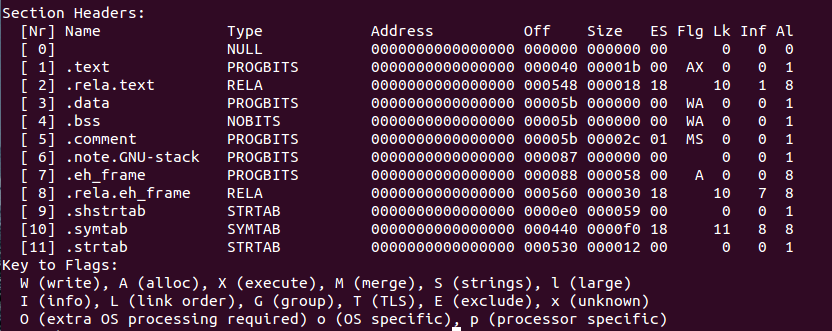


* **Section**

完成了对Header的解析，再接着分析Section部分，Section对应结构体如下：

|  |
| --- |
| typedef struct  {  Elf64\_Word sh\_name; /\* Section name (string tbl index) \*/  Elf64\_Word sh\_type; /\* Section type \*/  Elf64\_Xword sh\_flags; /\* Section flags \*/  Elf64\_Addr sh\_addr; /\* Section virtual addr at execution \*/  Elf64\_Off sh\_offset; /\* Section file offset \*/  Elf64\_Xword sh\_size; /\* Section size in bytes \*/  Elf64\_Word sh\_link; /\* Link to another section \*/  Elf64\_Word sh\_info; /\* Additional section information \*/  Elf64\_Xword sh\_addralign; /\* Section alignment \*/  Elf64\_Xword sh\_entsize; /\* Entry size if section holds table \*/  } Elf64\_Shdr; |

Section部分主要存放的是机器指令代码和数据，执行命令readelf -S -W hello.o对Section部分的解析，解析结果和Elf64\_Shdr也是一一对应的。



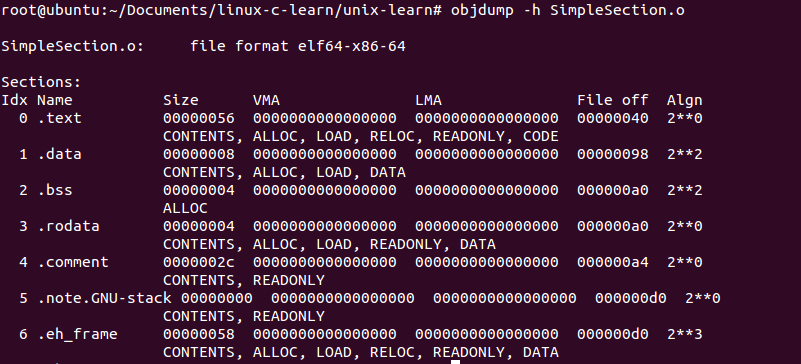
对于这部分内容，通常我们比较的Section是**.text（存放代码）**、**.data（存放全局静态变量和局部静态变量）**和**.bss（存未初始化的全局变量和局部静态变量）**，在后面会对这几个段分别分进行解析。

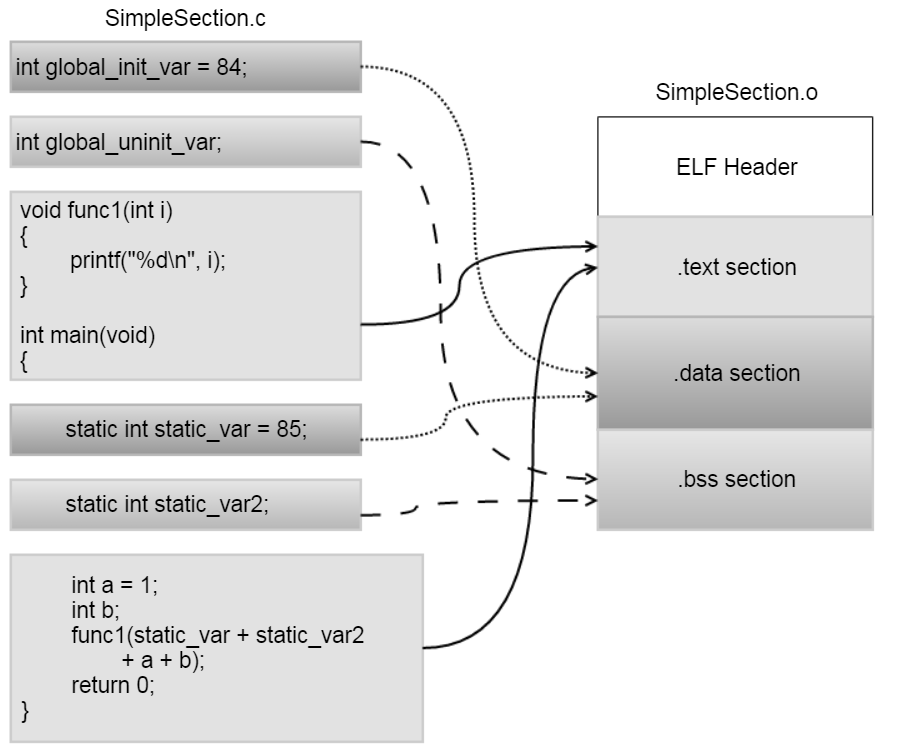
**解析目标文件**

分析完ELF文件结构，接着来解析一个目标文件。首先，准备好源码SimpleSection.c，执行命令gcc -c SimpleSection.c生成目标文件SimpleSection.o。

|  |
| --- |
| int printf(const char\* format, ...);  int global\_init\_var = 84;  int global\_uninit\_var;  void func1(int i)  {  printf("%d\n", i);  }  int main(void)  {  static int static\_var = 85;  static int static\_var2;  int a = 1;  int b;  func1(static\_var + static\_var2 + a + b);  return 0;  } |

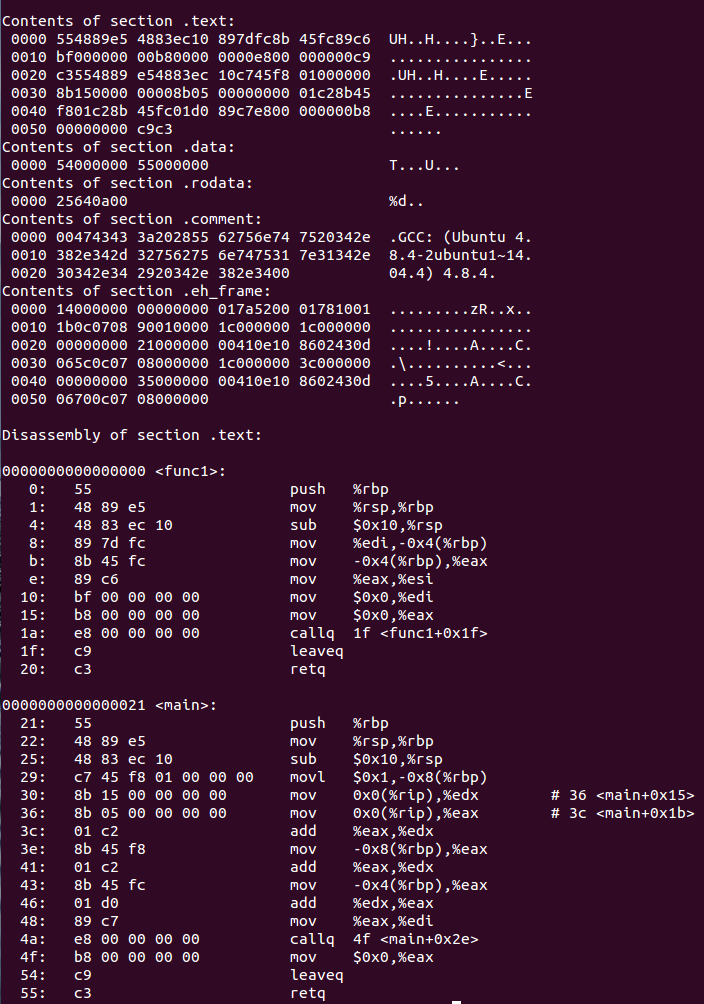
执行命令objdump -h SimpleSection.o对Section部分进行解析，我们可以得到每个段的大小。





我们的代码是存放到.text中，已初始化全局变量和局部静态变量存放在.data中，未初始化全局变量和局部静态变量存放在.bss中。

执行命令**objdump -s -d SimpleSection.o**对代码段(.text)的解析结果如下：



执行命令**objdump -s -d SimpleSection.o**对数据段和只读数据段解析结果如下：

执行命令objdump -x -s -d SimpleSection.o打印出目标文件的符号表：