dl\_iterate\_phdr

dl\_iterate\_phdr

**int dl\_iterate\_phdr(**

**int (\***callback**) (struct dl\_phdr\_info \***info**,**

**size\_t** size**, void \***data**),**

**void \***data**);**

The *info* argument is a structure of the following type:

struct dl\_phdr\_info {

ElfW(Addr) dlpi\_addr; /\* Base address of object \*/

const char \*dlpi\_name; /\* (Null-terminated) name of

object \*/

const ElfW(Phdr) \*dlpi\_phdr; /\* Pointer to array of

ELF program headers

for this object \*/

ElfW(Half) dlpi\_phnum; /\* # of items in *dlpi\_phdr* \*/

};

Note that we can calculate the location of a particular program header, *x*, in virtual memory using the formula:

addr == info->dlpi\_addr + info->dlpi\_phdr[x].p\_vaddr;

命令行工具必须的库：

1 配置解析 xml ,json

配置解析导出表hook和函数hook

struct ImportRefConfig{

std::string key;

std::string value;

}

std::list<ImportRefConfig> lst\_import\_hook

std::list<ElfHookConfig> lst\_elf\_hook

2 参数解析 void main(int argc, char\* argv[]) print\_help() gflags

重定位过程要用到一些表，来辅助定位，有.rel.text、 .rel.dyn和.rel.plt、.plt.下面根据他们来展开说明。  
  
.rel.text   
  重定位的地方在.text段内，以offset指定具体要定位位置。在连接时候由连接器完成。注意比较.text段前后变化。指的是比较.o文件和最终的执行文件（或者动态库文件）。就是重定位前后比较，以上是说明了具体比较对象而已。  
  
.rel.dyn   
  重定位的地方在.got段内。主要是针对外部数据变量符号。例如全局数据。重定位在程序运行时定位，一般是在.init段内。定位过程：获得符号对应value后，根据rel.dyn表中对应的offset，修改.got表对应位置的value。另外，.rel.dyn 含义是指和dyn有关，一般是指在程序运行时候，动态加载。区别于rel.plt，rel.plt是指和plt相关，具体是指在某个函数被调用时候加载。  
    
.rel.plt    
  重定位的地方在.got.plt段内（注意也是.got内,具体区分而已）。 主要是针对外部函数符号。一般是函数首次被调用时候重定位。可看汇编，理解其首次访问是如何重定位的，实际很简单，就是初次重定位函数地址，然后把最终函数地址放到.got.plt内，以后读取该.got.plt就直接得到最终函数地址(参考过程说明)。  所有外部函数调用都是经过一个对应桩函数，这些桩函数都在.plt段内。  
    
 过程说明:调用对应桩函数--->桩函数取出.got表(具体是.got.plt)表内地址--->然后跳转到这个地址.如果是第一次,这个跳转地址默认是桩函数本身跳转处地址的下一个指令地址(目的是通过桩函数统一集中取地址和加载地址),后续接着把对应函数的真实地址加载进来放到.got.plt表对应处,同时跳转执行该地址指令.以后桩函数从.got.plt取得地址都是真实函数地址了。  
     
 .plt段，存放重定位桩函数的。  
  
重要区别    
.rel.text属于普通重定位辅助段 ,他由编译器编译产生，存在于obj文件内。连接器连接时，他 用于最终可执行文件或者动态库的重定位。通过它修改原obj文件的.text段后，和并      到 最终可执行文件或者动态文件的.text段。  
*注：readelf -r a.o 查看 .rel.text。其类型一般为R\_386\_32和R\_386\_PC32*  
.rel.dyn和.rel.plt是动态定位辅助段。由连接器产生，存在于可执行文件或者动态库文件内。借助这两个辅助段可以动态修改对应.got和.got.plt段，从而实现运行时重定位。  
.rel.dyn 对应地点在.got表内；.rel.plt  在.got.plt,注意不是在.text，这点和普通不同，也是重要点。  
  
.rel.text由编译器产生，然后在连接时候，由链接器负责根据.rel.text对.text段进行修改，从而达到重定位目的；  
  
.rel.dyn和.rel.plt由连接器产生，然后在运行时候，动态加载符号地址。  
对于数据，根据.rel.dyn找到.got中的offset位置；  
对于函数则通过.plt桩函数和.rel.plt段来获取函数真实地址，然后存在于.got.plt。  
要理解动态连接中访问外部符号是通过.got和.got.plt  
*注1：规律：.rel.plt和.got.plt偏移 有对应，另外和.dynsym好似有次序对应关系(不很确定)。通过这可方便解析函数符号地址，详细可参考连接器解析函数（连接器一般是动态加载，像全局数据符号那样，程序启动时加载）。*

<http://nicephil.blinkenshell.org/my_book/index.html>

ELF32Reader

<template typename> C++ 模板，偏特化

C++ 类 初始化

C++ lamba 表达式

1 字符串表 .dynstr

2 符号表 .dynsym

ELF32\_Sym

typedef struct {  
      Elf32\_Word st\_name;  
      Elf32\_Addr st\_value;  
      Elf32\_Word st\_size;  
      unsigned char st\_info;  
      unsigned char st\_other;  
      Elf32\_Half st\_shndx;  
  } Elf32\_Sym;

**问题：为什么附属ELF除了导入函数符号信息外，其他都是无用的？**

Elf32\_Sym\* sym;

sym->st\_name += offset

sym->st\_size = 0

3 INIT\_ARRAY 数组

.init\_array section name

.rel.dyn 可能需要重定位

合并过程中需要保存 重定位信息

Elf32\_Rel

typedef struct{

Elf32\_Addr r\_offset;

Elf32\_Word r\_info;

} Elf32\_Rel;

4 处理重定位

.rel.plt

.rel.dyn

<http://linux.chinaunix.net/techdoc/system/2008/01/28/977690.shtml>

符号引用

符号定义

重定位类型：

R\_ARM\_JUMP\_SLOT

R\_ARM\_ABS32

R\_ARM\_GLOB\_DAT

R\_ARM\_RELATIVE

R\_386\_RELATIVE

R\_386\_GLOB\_DAT

R\_386\_32

R\_386\_JMP\_SLOT

5 .dynamic

DT\_INIT

与Dynamic Linking 有关的section

.dynamic

addresses of other dynamic linking information

data segment

.hash

1 symbol hash table

2 data segment

.plt

procedure linkage table

text segment

.got

global offset table

data segment

ELF execution view

Program Header Table

Text Segment read-only

.text .rodata .hash .dynsym .synstr .plt .rel.got

Data Segment writable data

.data .dynamic .got .bss

BSS Segment .bss section

**Program Header Table**

/\* Program segment header. \*/

typedef struct

{

Elf32\_Word p\_type; /\* Segment type \*/

Elf32\_Off p\_offset; /\* Segment file offset \*/

Elf32\_Addr p\_vaddr; /\* Segment virtual address \*/

Elf32\_Addr p\_paddr; /\* Segment physical address \*/

Elf32\_Word p\_filesz; /\* Segment size in file \*/

Elf32\_Word p\_memsz; /\* Segment size in memory \*/

Elf32\_Word p\_flags; /\* Segment flags \*/

Elf32\_Word p\_align; /\* Segment alignment \*/

} Elf32\_Phdr;

**process\_vm\_readv**