objdump的使用(RedHat6.0测试)

发布日期：2000-01-18

文摘内容：

发信人: cloudsky (小四), 信区: Security

标 题: objdump的使用(RedHat6.0测试)

发信站: 武汉白云黄鹤站 (Wed Jan 12 16:30:30 2000), 站内信件

标题：objdump的使用(RedHat6.0测试)

这个混蛋躺在你主机上很久了，只是你不看而已，

可能不是你不想看，而是没时间琢磨，那我就琢磨

出来叫你省点事，faint

概述：

objdump有点象那个快速查看之流的工具，就是

以一种可阅读的格式让你更多地了解二进制文件

可能带有的附加信息。对于一般只想让自己程序

跑起来的程序员，这个命令没有更多意义，对于

想进一步了解系统的程序员，应该掌握这种工具，

至少你可以自己写写shellcode了，或者看看人家

给的exploit中的shellcode是什么东西。

目录：

★ 测试练习前的准备工作

cp /usr/lib/libpcap.a /home/scz/src

nm -s libpcap.a | more

ar tv libpcap.a

ar xv libpcap.a inet.o

nm -s inet.o

关于nm -s的显示请自己man nm查看

★ Redhat 6.0 objdump命令的man手册

objdump - 显示二进制文件信息

objdump

[-a] [-b bfdname |

--target=bfdname] [-C] [--debugging]

[-d] [-D]

[--disassemble-zeroes]

[-EB|-EL|--endian={big|little}] [-f]

[-h] [-i|--info]

[-j section | --section=section]

[-l] [-m machine] [--prefix-addresses]

[-r] [-R]

[-s|--full-contents] [-S|--source]

[--[no-]show-raw-insn] [--stabs] [-t]

[-T] [-x]

[--start-address=address] [--stop-address=address]

[--adjust-vma=offset] [--version] [--help]

objfile...

--archive-headers

-a 显示档案库的成员信息，与 ar tv 类似

objdump -a libpcap.a

和 ar -tv libpcap.a 显示结果比较比较

显然这个选项没有什么意思。

--adjust-vma=offset

When dumping information, first add offset to all the section addresses. This is useful if the section addresses do not correspond to the symbol table, which can happen when putting sections at particular addresses when using a format which can not represent section addresses, such as a.out.

-b bfdname

--target=bfdname

指定目标码格式。这不是必须的，objdump能自动识别许多格式，

比如：objdump -b oasys -m vax -h fu.o

显示fu.o的头部摘要信息，明确指出该文件是Vax系统下用Oasys

编译器生成的目标文件。objdump -i将给出这里可以指定的

目标码格式列表

--demangle

-C 将底层的符号名解码成用户级名字，除了去掉所有开头

的下划线之外，还使得C++函数名以可理解的方式显示出来。

--debugging

显示调试信息。企图解析保存在文件中的调试信息并以C语言

的语法显示出来。仅仅支持某些类型的调试信息。

--disassemble

-d 反汇编那些应该还有指令机器码的section

--disassemble-all

-D 与 -d 类似，但反汇编所有section

--prefix-addresses

反汇编的时候，显示每一行的完整地址。这是一种比较老的反汇编格式。

显示效果并不理想，但可能会用到其中的某些显示，自己可以对比。

--disassemble-zeroes

一般反汇编输出将省略大块的零，该选项使得这些零块也被反汇编。

-EB

-EL

--endian={big|little}

这个选项将影响反汇编出来的指令。

little-endian就是我们当年在dos下玩汇编的时候常说的高位在高地址，

x86都是这种。

--file-headers

-f 显示objfile中每个文件的整体头部摘要信息。

--section-headers

--headers

-h 显示目标文件各个section的头部摘要信息。

--help 简短的帮助信息。

--info

-i 显示对于 -b 或者 -m 选项可用的架构和目标格式列表。

--section=name

-j name 仅仅显示指定section的信息

--line-numbers

-l 用文件名和行号标注相应的目标代码，仅仅和-d、-D或者-r一起使用

使用-ld和使用-d的区别不是很大，在源码级调试的时候有用，要求

编译时使用了-g之类的调试编译选项。

--architecture=machine

-m machine

指定反汇编目标文件时使用的架构，当待反汇编文件本身没有描述

架构信息的时候(比如S-records)，这个选项很有用。可以用-i选项

列出这里能够指定的架构

--reloc

-r 显示文件的重定位入口。如果和-d或者-D一起使用，重定位部分以反汇

编后的格式显示出来。

--dynamic-reloc

-R 显示文件的动态重定位入口，仅仅对于动态目标文件有意义，比如某些

共享库。

--full-contents

-s 显示指定section的完整内容。

objdump --section=.text -s inet.o | more

--source

-S 尽可能反汇编出源代码，尤其当编译的时候指定了-g这种调试参数时，

效果比较明显。隐含了-d参数。

--show-raw-insn

反汇编的时候，显示每条汇编指令对应的机器码，除非指定了

--prefix-addresses，这将是缺省选项。

--no-show-raw-insn

反汇编时，不显示汇编指令的机器码，这是指定 --prefix-addresses

选项时的缺省设置。

--stabs

Display the contents of the .stab, .stab.index, and

.stab.excl sections from an ELF file. This is only

useful on systems (such as Solaris 2.0) in which

.stab debugging symbol-table entries are carried in

an ELF section. In most other file formats, debug-

ging symbol-table entries are interleaved with

linkage symbols, and are visible in the --syms output.

--start-address=address

从指定地址开始显示数据，该选项影响-d、-r和-s选项的输出。

--stop-address=address

显示数据直到指定地址为止，该选项影响-d、-r和-s选项的输出。

--syms

-t 显示文件的符号表入口。类似于nm -s提供的信息

--dynamic-syms

-T 显示文件的动态符号表入口，仅仅对动态目标文件有意义，比如某些

共享库。它显示的信息类似于 nm -D|--dynamic 显示的信息。

--version 版本信息

objdump --version

--all-headers

-x 显示所有可用的头信息，包括符号表、重定位入口。-x 等价于

-a -f -h -r -t 同时指定。

objdump -x inet.o

参看 nm(1)

★ objdump应用举例(待增加)

/*

```
g++ -g -Wstrict-prototypes -Wall -Wunused -o objtest objtest.c
*/
#include
#include
int main ( int argc, char * argv[] )
{
    execl( "/bin/sh", "/bin/sh", "-i", 0 );
    return 0;
}
g++ -g -Wstrict-prototypes -Wall -Wunused -o objtest objtest.c
objdump -j .text -Sl objtest | more
/main(查找)
08048750 :
main():
/home/scz/src/objtest.c:7
*/
#include
#include
int main ( int argc, char * argv[] )
{
8048750:       55                    pushl  %ebp
8048751:       89 e5                 movl   %esp,%ebp
/home/scz/src/objtest.c:8
        execl( "/bin/sh", "/bin/sh", "-i", 0 );
8048753:       6a 00                 pushl  $0x0
8048755:       68 d0 87 04 08          pushl  $0x80487d0
804875a:       68 d3 87 04 08          pushl  $0x80487d3
804875f:       68 d3 87 04 08          pushl  $0x80487d3
8048764:       e8 db fe ff ff          call   8048644 <_init+0x40>
8048769:       83 c4 10              addl   $0x10,%esp
/home/scz/src/objtest.c:9
        return 0;
804876c:       31 c0                 xorl   %eax,%eax
804876e:       eb 04                 jmp    8048774
```

```
8048770:       31 c0                xorl   %eax,%eax
8048772:       eb 00                jmp    8048774
/home/scz/src/objtest.c:10
}
8048774:       c9                   leave
8048775:       c3                   ret
8048776:       90                   nop
```

如果说上面还不够清楚，可以用下面的命令辅助一下：

```
objdump -j .text -Sl objtest --prefix-addresses | more
objdump -j .text -Dl objtest | more
```

去掉调试编译选项重新编译

```
g++ -O3 -o objtest objtest.c
objdump -j .text -S objtest | more
08048778 :
main():
8048778:       55                   pushl  %ebp
8048779:       89 e5                movl   %esp,%ebp
804877b:       6a 00                pushl  $0x0
804877d:       68 f0 87 04 08          pushl  $0x80487f0
8048782:       68 f3 87 04 08          pushl  $0x80487f3
8048787:       68 f3 87 04 08          pushl  $0x80487f3
804878c:       e8 db fe ff ff          call   804866c <_init+0x40>
8048791:       31 c0                xorl   %eax,%eax
8048793:       c9                   leave
8048794:       c3                   ret
8048795:       90                   nop
```

与前面-g编译后的二进制代码比较一下，有不少区别。

至于如何写shellcode、如何理解别人给出的shellcode，请参看华中站

系统安全版精华区中的"如何写自己的shellcode"

★ 相关命令

```
man objcopy
man nm
```

man gdb | dbx | sdb

gdb的中文使用使用手册在各大高校BBS精华区中都有，自己去看，如果你

想搞点名堂，是应该学习使用这个工具。