逆向与反汇编工具

转载

felixs 于 2013-04-09 14:17:29 发布 3391 收藏
分类专栏： 技术 反汇编

技术 同时被 2 个专栏收录
245 篇文章 0 订阅
订阅专栏

反汇编
4 篇文章 0 订阅
订阅专栏

```
// Dump the generated code in an asm file format that can be assembled and then disassembled
// for debugging purposes. For example, save this output as jit.s:
```

```
1. //   gcc -c jit.s
   //   objdump -D jit.o
```

```
2.     //   gcc -arch armv7 -c jit.s
   //   otool -tv jit.o
```

```
3. //   shasm -arch
```

转自：http://blog.163.com/shanshenye2k@yeah/blog/static/823405412012930555115/

反汇编程序一般有两种，静态反汇编如W32Dasm,动态反汇编的如Softice.

IDA.Pro参考网址：http://www.doc88.com/p-739479754476.html

第 1 章　　逆向与反汇编工具

了解反汇编的一些背景知识后，再深入学习*IDA Pro*之前，介绍其他一些用于二进制文件的逆向工程工具，会对我们学习有所帮助。这些工具大多在*IDA*之前发布，并且仍然可用于快速分析二进制文件，以及审查*IDA*的分析结果。如我们所见，*IDA*将这些工具的诸多功能整合到它的用户界面中，为逆向工程提供了一个集成环境。最后，尽管*IDA*确实包含一个集成调试器，在这里我们不会讨论，因为在第*24*、*25*和*26*章专门讨论这个主题。

1.1　分类工具

通常，第一次面对一个未知文件时，有必要问一些简单问题是有益的，如“这是个什么东西？”回答这个问题的首要原则，是不要依赖文件扩展名来确定文件的类型。这是最基本的原则。在脑子里建立起“文件扩展名并无实际意义”的印象后，你就会开始考虑学习下面几个实用工具。

1.1.1　*file*

*file*命令是一个标准的实用工具，大多数**NIX*风格的操作系统和*Windows*下的*Cygwin[1]*或*MinGW[2]*工具都带有这个实用工具。*file*试图通过检查文件中某些特定字段来确定文件类型。在某些情况下，*file*能够识别常见的字符串，如*#!/bin/sh*（*shell*脚本文件）或*<html>*（*HTML*文档）。但是，识别那些包含非*ASCII*内容的文件要困难得多，在这种情况下，*file*会设法判断，该文件的结构是否符合某种已知的文件格式。多数情况下，它会搜索某些文件类型所特有的标签值（通常称为幻数*[3]*）。下面的十六进制表列出了几个用于判断常见文件类型的幻数。

Windows PE executable file

00000000　4D 5A 90 00 03 00 00 00　04 00 00 00 FF FF 00 00　MZ..............

00000010 B8 00 00 00 00 00 00 00　40 00 00 00 00 00 00 00　........@.......

Jpeg image file

00000000 FF D8 FF E0 00 10 4A 46　49 46 00 01 01 01 00 60　......JFIF.....`

00000010 00 60 00 00 FF DB 00 43　00 0A 07 07 08 07 06 0A　.`.....C........

Java .class file

00000000 CA FE BA BE 00 00 00 32　00 98 0A 00 2E 00 3E 08　.......2......>.

00000010 00 3F 09 00 40 00 41 08　00 42 0A 00 43 00 44 0A　.?..@.A..B..C.D.

---

*file*具有识别大量的文件格式的能力，包括多种类型的*ASCII*文本文件、可执行文件和数据文件格式。*file*执行的幻数检查由幻数文件（*magic file*）所包含的规则控制。幻数文件的默认位置因操作系统而异，常见的位置包包*/usr/share/file/magic*、*/usr/share/misc/magic*和*/edt/magic*。关于幻数文件更多的信息，请参阅*file*的文挡资料。

*CYGWIN*环境

*Cygwin*是*Windows*操作系统中的一组实用工具，可提供*Linux*风格的命令行解释器（*command shell*）和相关程序。在安装过程中，有大量安装包可供用户选择，包括编译器（*gcc*、*g++*）；解释器（*Perl*、*Python*、*Ruby*）；网络实用工具（*nc*、*ssh*）等等。*Cygwin*安装完毕，许多为*Linux*编写的程序就可以在*Windows*系统中编译和执行。

在某些情况下，*file*还能够辨别某一指定文件类型中的细微变化。以下列表证实了*file*不仅能够识别几种不同的*ELF*二进制文件，而且还提供了有关二进制文件如何链接（静态或动态）以及是否去除了符号等信息。

```
idabook# file ch2_ex_*

ch2_ex.exe:          MS-DOS executable PE for MS Windows (console)
                        Intel 80386 32-bit

ch2_ex_upx.exe:      MS-DOS executable PE for MS Windows (console)
                        Intel 80386 32-bit, UPX compressed

ch2_ex_freebsd:      ELF 32-bit LSB executable, Intel 80386,
                        version 1 (FreeBSD), for FreeBSD 5.4,
                        dynamically linked (uses shared libs),
                        FreeBSD-style, not stripped

ch2_ex_freebsd_static:    ELF 32-bit LSB executable, Intel 80386,
                        version 1 (FreeBSD), for FreeBSD 5.4,
                        statically linked, FreeBSD-style, not stripped

ch2_ex_freebsd_static_strip: ELF 32-bit LSB executable, Intel 80386,
                        version 1 (FreeBSD), for FreeBSD 5.4,
                        statically linked, FreeBSD-style, stripped
```

ch2_ex_linux: ELF 32-bit LSB executable, Intel 80386,

version 1 (SYSV), for GNU/Linux 2.6.9,

dynamically linked (uses shared libs),

not stripped

ch2_ex_linux_static: ELF 32-bit LSB executable, Intel 80386,

version 1 (SYSV), for GNU/Linux 2.6.9,

statically linked, not stripped

ch2_ex_linux_static_strip: ELF 32-bit LSB executable, Intel 80386,

version 1 (SYSV), for GNU/Linux 2.6.9,

statically linked, stripped

ch2_ex_linux_stripped: ELF 32-bit LSB executable, Intel 80386,

version 1 (SYSV), for GNU/Linux 2.6.9,

dynamically linked (uses shared libs), stripped

---

*file*及类似的实用工具同样也会出错。如果一个文件包含某些文件格式的标志，这些工具很可能会产生误判。你可以使用一个十六进制文件编辑器将任何文件的前*4*个字节修改为*Java*的幻数序列：*CA FE BA BE*，自己证实一下上述情况。这时，*file*会将这个新修改的文件错误地识别为*"*已编译的*Java*类数据*"*。同样，一个只包含*MZ*这两个字符的文本文件会被误认为是一个*MS-DOS*可执行文件。在逆向工程过程中，一个良好的习惯是，绝不要完全相信任何工具所提供的结果，除非该结果得到其他几款工具和手动分析的确认。

1.1.2 PE Tools

*PE Tools[4]*是一个实用工具的集合，用于分析*Windows*系统中正在运行的进程和可执行文件。*PE Tools*的主界面如图*2-1*所示，其中列出了所有活动进程，并提供所有的*PE Tools*实用工具。

图*2-1*：*PE Tools*实用程序

模糊是指任何企图掩盖真正意义上的东西。当应用到可执行文件，模糊是指任何试图隐藏程序的真实行为。有许多原因可以让程序员对程序采用模糊处理。普遍引用的例子包括：保护专有算法和模糊恶意意图。几乎所有恶意软件的形式利用模糊处理，以阻碍对其进行分析。有大量模糊工具可供程序员使用，帮助他们创建模糊程序。模糊处理工具和技术，以及对逆向工程过程的相关影响，将在第*21*章中进一步讨论。

在进程列表中，用户可以将进程的内存映像转储到某个文件，或利用PE Sniffer实用工具确定可执行文件由何种编译器构建，或者该文件是否经过某种已知的模糊处理实用工具处理。Tools菜单提供了磁盘文件分析的类似选项。另外，用户还可以使用内嵌的PE Editor实用工具查看PE文件头字段，使用该工具可以方便修改任何文件头的值。通常，如果想要从一个文件的模糊版本重建一个有效的PE，就需要修改PE文件头。

1.1.3 PEiD

PEiD[5]是另一款Windows工具，它主要用于识别构建某一特定Windows PE二进制文件所使用的编译器，并确定任何用于模糊Windows PE二进制文件的工具。图2-2显示了如何使用PEiD确定模糊Gaobot[6]蠕虫的一个变种所使用的工具（此例中为ASPack）。

图2-2：PEiD实用工具

PEiD的许多其他功能与PETools的功能相同，包括显示PE文件头信息摘要、收集有关正在运行的进程的信息、执行基本的反汇编等。

1.2 摘要工具

由于我们的目标是对二进制程序文件进行逆向工程，因此，在对文件进行初步分类后，需要用更高级的工具来提取详尽的信息。本节讨论的工具不只能识别它们所处理的文件的格式，更重要的是还能够理解某一特定的文件格式，并且能够解析它们的输入文件，提取出这些输入文件所包含的非常特别的信息。

1.2.1 nm

当源文件编译为目标文件，编译器必须嵌入一些全局（外部）符号的位置信息，以便链接器在组合目标文件以创建可执行文件时，能够解析对这些符号的引用。除非被告知要去除最终的可执行文件中的符号，否则，链接器通常会将目标文件中的符号带入最终的可执行文件中。根据nm手册的描述，这一实用工具的目的是"列举目标文件中的符号"。

使用nm检查中间目标文件（扩展名为.o的文件，而非可执行文件）时，默认输出结果是在这个文件中声明的任何函数和全局变量的名称。nm实用工具的样本输出如下所示：

```
idabook# gcc -c ch2_example.c

idabook# nm ch2_example.o

         U __stderrp

         U exit
```

U fprintf

00000038 T get_max

00000000 t hidden

00000088 T main

00000000 D my_initialized_global

00000004 C my_unitialized_global

U printf

U rand

U scanf

U srand

U time

00000010 T usage idabook#

从中可以看到，nm列出了每个符号以及与符号有关的一些信息。其中的字母表示所列举符号的类型。这里我们解释前面的例子中出现了以下字母代码：

–未定义符号，通常为外部符号引用。
-在文本部分定义的符号，通常为函数名称。
在文本部分定义的局部符号。在C程序中，这个符号通常等同于一个静态函数。
–已初始化的数据值。
–未初始化的数据值。

大写字母表示全局符号，小写字母则表示局部符号。请参阅nm手册了解有关字母代码的详细解释。

使用nm列举可执行文件中的符号，会有更多信息显示出来。在链接过程中，符号被解析成虚拟地址（如有可能）。因此，这时运行nm，将可获得更多信息。下面是使用nm处理一个可执行文件得到的部分输出：

idabook# gcc -o ch2_example ch2_example.c

idabook# nm ch2_example

<. . .>

U exit

U fprintf

080485c0 t frame_dummy

08048644 T get_max

0804860c t hidden

08048694 T main

0804997c D my_initialized_global

08049a9c B my_unitialized_global

08049a80 b object.2

08049978 d p.0

U printf

U rand

U scanf

U srand

U time

0804861c T usage

idabook#

在这个例子中，一些符号(如main)分配了虚拟地址，链接过程引入了一些新的符号(如frame_dummy)，另一些符号(如my_unitialized_global)的类型发生了改变，其他符号由于继续引用外部符号，仍旧为未定义符号。在这个特例中，我们检测的文件属于动态链接二进制文件，为此，未定义的符号将在C语言共享库中定义。要了解更多有关nm的信息，请参阅nm手册。

1.2.2 ldd

创建可执行文件时，必须解析该文件引用的任何库函数的地址。链接器通过两种方法解析对库函数的调用：静态链接(static linking)和动态链接(dynamic linking)。链接器的命令行参数决定具体使用哪一种方法。一个可执行文件可能为静态链接、动态链接，或二者兼而有之[7]。

要求静态链接时，链接器会将程序的目标文件和所需的库文件组合起来，生成一个可执行文件。这样，在运行时就不需要确定库代码的位置，因为它已经包含在可执行文件中。静态链接的优点：（1）函数调用速度会更快些；（2）发布二进制文件更容易，因为不需要对用户系统中库函数的可用性做出任何假设。缺点包括：（1）生成的可执行文件较大；（2）如果库组件发生改变，对程序进行升级会更加困难，因为一旦库发生变化，程序就必须重新链接。从逆向工程的角度看，静态链接使问题更加复杂。在分析一个静态链接二进制文件时，要回答"这个二进制文件链接了哪些库"，可不是那么容易。我们将在第12章讨论在对静态链接代码进行逆向工程时遇到的挑战。

动态链接与静态链接不同。使用动态链接时，链接器不需要复制它需要的任何库。相反，链接器只需将对所需库（通常为.so或.dl文件）的引用插入到最终的可执行文件中。因此，生成的可执行文件也会更小些。而且，使用动态链按时升级库代码也变得简单多了，因为只需要维护一个库（被许多二进制文件引用）。如果需要升级库代码，用新版本的库替换过时的库，就可以立即更新每一个引用该库的二进制文件。使用动态链接的一个缺点是，它需要更复杂的加载过程。因为这时必须定位所有所需的库，并将其加载到内存中，而不是加载一个包含全部库代码的静态链接文件。动态链接的另一个缺点，是供应商不仅需要发布他们自己的可执行文件，而且必须发布该文件所需的所有库文件。如果一个系统无法提供程序所需的全部库文件，在这个系统上运行该程序将会导致错误。

下面的输出说明了一个程序的动态和静态链接版本的创建过程、生成的二进制文件的大小，以及如何使用file工具识别这两个程序：

---

idabook# gcc -o ch2_example_dynamic ch2_example.c

idabook# gcc -o ch2_example_static ch2_example.c --static

*idabook# ls -l ch2_example_**

-rwxr-xr-x 1 root wheel 6017 Sep 26 11:24 ch2_example_dynamic

-rwxr-xr-x 1 root wheel 167987 Sep 26 11:23 ch2_example_static

*idabook# file ch2_example_**

ch2_example_dynamic: ELF 32-bit LSB executable, Intel 80386, version 1

(FreeBSD), dynamically linked (uses shared libs), not stripped

ch2_example_static: ELF 32-bit LSB executable, Intel 80386, version 1

(FreeBSD), statically linked, not stripped

idabook#

为了确保动态链接正常运行，动态链接二进制文件必须指明它需要的库文件，以及需要这些文件中的哪些特定资源。因此，与静态链接二进制文件不同，我们可轻易确定一个动态链接二进制文件所依赖的库文件。ldd (list dynamic dependencies) 是一个简单的实用工具，可用来列举任何可执行文件所需的动态库。在下面这个例子中，我们使用ldd确定Apache Web服务器所依赖的库：

---

idabook# ldd /usr/local/sbin/httpd

/usr/local/sbin/httpd:

libm.so.4 => /lib/libm.so.4 (0x280c5000)

libaprutil-1.so.2 => /usr/local/lib/libaprutil-1.so.2 (0x280db000)

libexpat.so.6 => /usr/local/lib/libexpat.so.6 (0x280ef000)

libiconv.so.3 => /usr/local/lib/libiconv.so.3 (0x2810d000)

libapr-1.so.2 => /usr/local/lib/libapr-1.so.2 (0x281fa000)

```
        libcrypt.so.3 => /lib/libcrypt.so.3 (0x2821a000)

        libpthread.so.2 => /lib/libpthread.so.2 (0x28232000)

        libc.so.6 => /lib/libc.so.6  (0x28257000)

idabook#
```

---

ldd实用工具可用于Linux和BSD系统。在OS X系统上，使用otool工具，并带上-L选项(otool -L 文件名)，即可实现类似的功能。在Windows系统中，可以使用Visual Studio工具套件中的实用工具dumpbin列举某文件所依赖的库，形式为：dumpbin /dependents 文件名。

1.2.3 objdump

ldd相当专业，而objdump非常灵活。objdump的主要目的是“显示目标文件中的信息。”[8]。这是一个相当广泛的目标，objdump为此提供了大量命令行选项（超过30个），以提取目标文件中的各种信息。objdump可用于显示以下与目标文件相关的数据（以及其他更多信息）：

节头（Section headers）

在程序文件中的每一节的摘要信息。

私有头（Private headers）

程序存储器的布局信息以及运行时加载器所需的其他信息，包括由ldd等工具生成的库列表。

调试信息（Debugging information）

提取出嵌入在程序文件中的任何调试信息。

符号信息（Symbol information）

以类似nm的方式转储符号表信息。

反汇编列表（Disassembly listing）

objdump对文件中标记为代码的部分执行线性扫描反汇编。反汇编x86代码时，objdump可以生成AT&T或Intel语法，并可以将反汇编代码保存在文本文件中。这样的文本文件叫做反汇编完全列表(dead listing)，尽管这些文件可用于实施逆向工程，但它们很难有效导航，也无法以一致且无错的方式修改。

objdump是GNU binutils[9]工具套件的一部分，用户可以在Linux、FreeBSD和Windows（通过Cygwin）系统中找到这个工具。objdump依靠二进制文件描述符库libbfd（二进制工具的一个组件）来访问目标文件，因此，它能够解析libbfd支持的文件格式（ELF、PE等）。另外，一个名为readelf的实用工具也可用于解析ELF文件。readelf的大多数功能与objdump相同，它们之间的主要区别在于：readelf并不依赖libbfd。

1.2.4 otool

otool可用于解析与OS X Mach-O二进制文件有关的信息，因此，可简单将其描述为：OS X系统下的类似于objdump的实用工具。下面的代码说明了如何使用otool显示一个Mach-O二进制文件的动态库依赖关系，从而执行类似于ldd的功能。

```
idabook# file osx_example
```

osx_example: Mach-O executable ppc

idabook# otool -L osx_example

osx_example:

/usr/lib/libstdc++.6.dylib (compatibility version 7.0.0, current version 7.4.0)

/usr/lib/libgcc_s.1.dylib (compatibility version 1.0.0, current version 1.0.0)

/usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 88.1.5)

---

otool可用于显示与文件的头和符号表有关的信息，并对文件的代码部分进行反汇编。了解更多有关otool功能的信息，请参阅相关手册。

1.2.5 dumpbin

dumpbin是微软Visual Studio工具套件中的一个命令行实用工具。与otool和objdump一样，dumpbin可以显示大量与Windows PE文件有关的信息。下面的例子说明了如何使用dumpbin以类似于ldd的方式显示Windows计算器程序的动态依赖关系。

$ dumpbin /dependents calc.exe

Microsoft (R) COFF/PE Dumper Version 8.00.50727.762

Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file calc.exe

File Type: EXECUTABLE IMAGE

Image has the following dependencies:

SHELL32.dll

msvcrt.dll

ADVAPI32.dll

KERNEL32.dll

GDI32.dll

USER32.dll

---

dumpbin的其他选项可从PE二进制文件的各个部分提取信息，包括符号、导入的函数名、导出的函数名和反汇编代码。要了解更多有关如何使用dumpbin的信息，请访问Mircrosoft Developer Network(MSDN)[10]。

1.2.6 c++filt

由于每个重载函数都使用与原函数相同的名称，因此，支持函数重载的语言必须拥有一种机制，以区分同一个函数的许多重载版本。下面的C++实例展示了一个名为demo的函数的几个重载版本的原型：

```
void demo(void);

void demo(int x);

void demo(double x);

void demo(int x, double y);

void demo(double x, int y);

void demo(char* str);
```

作为一般原则，一个目标文件中不可能有两个名称相同的函数。为允许重载，编译器将描述函数参数类型的信息合并到函数的原始名称中，生成重载函数的唯一名称。为名称完全相同的函数生成唯一名称的过程称为名称修饰(name mangling)。如果使用nm转储前面的C++代码的已编译版本中的符号，将得到如下结果（在demo版本的过滤焦点）：

```
idabook# g++ -o cpp_test cpp_test.cpp

idabook# nm cpp_test | grep demo

0804843c T _Z4demoPc

08048400 T _Z4demod

08048428 T _Z4demodi

080483fa T _Z4demoi

08048414 T _Z4demoid

080483f4 T _Z4demov
```

C++标准没有为名称改编方案制定标准，因此，编译器设计人员必须自己制定标准。为了译解上面列出的demo函数的重载版本，我们需要一个能够理解编译器（这里为g++）的名称改编方案的工具，c++filt正是这样一个实用工具。c++filt将每个输入的名称看成是改编后的名称(mangled name)[11]，并设法确定用于生成该名称的编译器。如果这个名称是一个合法的改编名称，那么，c++filt就输出改编之前的名称； 如果c++filt无法识别一个改编名称，那它就按原样输出该名称。

如果柴将上面nm输出的结果交给c++filt处理，它可以得到这些函数的原始名称，如下所示：

```
idabook# nm cpp_test | grep demo | c++filt

0804843c T demo(char*)

08048400 T demo(double)

08048428 T demo(double, int)

080483fa T demo(int)
```

08048414 T demo(int, double)

080483f4 T demo()

值得注意的是，改编名称可能包含其他与函数有关的信息，正常情况下，nm无法显示这些信息。在逆向工程过程中，这些信息可能非常重要。在更复杂的情况下，这些附加信息中可能还包含与类名称或函数调用约定有关的信息。

1.3 深度检测工具

到目前为止，我们已经讨论了一些工具，利用这些工具，可以在对文件的内部结构知之甚少的情况下对文件进行粗略分析，也可以在深入了解文件的结构之后，从文件中提取出特定的信息。在本节中，我们将介绍一些专用于从任何格式的文件中提取出特定信息的工具。

1.3.1 strings

有时候，提出一些与文件内容有关的常规性问题，即那些不需要了解文件结构即可回答的问题，对我们会有一定帮助。例如：“这个文件包含任何嵌入的字符串吗？”当然，在回答这个问题之前，必须先回答这个问题：“究竟是什么构成一个字符串？”我们将字符串简单定义为由可打印字符组成的连续字符序列。通常，在这一定义的基础上，还需要指定一个最小长度和一个特定的字符集。因此，可以搜索至少包含4个连续可打印ASCII字符的字符串，并将结果在控制台打印出来。搜索这类字符串一般不会受到文件结构的限制。在ELF二进制文件中搜索字符串就像在微软Word文挡中搜索字符串一样简单。

strings实用工具专门用于提取文件中的字符串内容，通常，使用该工具不会受到文件格式的限制。使用strings的默认设置（至少包含4个字符的7位ASCII序列），可得到以下结果：

---

idabook# strings ch2_example

/lib/ld-linux.so.2

gmon_start

libc.so.6

_IO_stdin_used

exit

srand

puts

time

printf

stderr

fwrite

scanf

libc_start_main

GLIBC_2.0

PTRh

[^_]

usage: ch2_example [max]

A simple guessing game!

Please guess a number between 1 and %d.

Invalid input, quitting!

Congratulations, you got it in %d attempt(s)!

Sorry too low, please try again

Sorry too high, please try again

不过，我们发现，一些字符串看起来像程序输出，一些字符串则像函数名称或库名称。因此，绝不能只根据这些字符串来断定程序的功能。分析人员往往会掉入陷阱，根据strings的输出来推断程序的功能。需要记住的是：二进制文件中包含某个字符串，并不表示该文件会以某种方式使用这个字符串。

下面是使用strings时需要注意的事项：

请记住：使用strings处理可执行文件时，默认情况下，strings只扫描文件中可加载的、经初始化的部分。使用命令行参数-a可迫使strings扫描整个文件。

strings不指出字符串在文件中的位置。使用-t命令行参数可使strings显示所发现的每一个字符串的文件偏移量信息。

许多文件使用了其他字符集。利用-e命令行参数可使strings搜索更广泛的字符，如16位Unicode字符。

1.3.2 反汇编器

如前所述，有很多工具都可以生成二进制目标文件的完全列表形式的反汇编。PE、ELF和MACH-O文件分别使用dumpbin、objdump和otool进行反汇编。但是，它们中任何一个都无法处理任意格式的二进制文件。有时候，你会遇到一些并不采用常用文件格式的二进制文件，在这种情况下，你就需要一些能够从用户指定的偏移量开始反汇编过程的工具。

两个用于x86指令集流式反汇编器：ndisasm和diStorm[12]。ndisasm是包含在Netwide Assembler(NASM)[13]中的一个实用程序。下面的例子说明了如何使用ndisasm反汇编一段由Metasploit框架[14]生成的shellcode：

```
idabook# ./msfpayload linux/x86/shell_findport CPORT=4444 R > fs

idabook# ls -l fs

-rw-r--r-- 1 ida ida 62 Dec 11 15:49 fs

idabook# ndisasm -u fs

00000000 31D2       xor edx,edx

00000002 52         push edx

00000003 89E5       mov ebp,esp

00000005 6A07       push byte +0x7

00000007 5B         pop ebx

00000008 6A10       push byte +0x10
```

```
0000000A  54              push esp

0000000B  55              push ebp

0000000C  52              push edx

0000000D  89E1        mov  ecx,esp

0000000F  FF01        inc  dword [ecx]

00000011  6A66        push byte  +0x66

00000013  58              pop eax

00000014  CD80        int 0x80

00000016  66817D02115C    cmp  word [ebp+0x2],0x5c11

0000001C  75F1        jnz  0xf

0000001E  5B              pop ebx

0000001F  6A02        push byte  +0x2

00000021  59              pop ecx

00000022  B03F        mov  al,0x3f

00000024  CD80        int 0x80

00000026  49              dec ecx

00000027  79F9        jns  0x22
```

```
00000029  52              push edx

0000002A  682F2F7368      push dword 0x68732f2f

0000002F  682F62696E      push dword 0x6e69622f

00000034  89E3            mov  ebx,esp

00000036  52              push edx

00000037  53              push ebx

00000038  89E1            mov  ecx,esp

0000003A  B00B            mov  al,0xb

0000003C  CD80            int 0x80
```

由于流式反汇编非常灵活，因此它的用途相当广泛。例如，在分析网络数据包中可能包含shellcode的计算机网络攻击时，就可采用流式反汇编器来反汇编数据包中包含shellcode的部分，分析恶意负载的行为。另外一种情况是分析那些格式未知的ROM镜像。ROM中有些部分是数据，其他部分则为代码，可以使用流式反汇编器来反汇编镜像中的代码。

1.4 小结

本章所讨论的工具不一定是同类中最好的，但它们是从事二进制文件逆向工程的分析人员常用的工具。更重要的是，这些工具大大促进了IDA的开发过程。在接下来的几章中，我们还会讨论这些工具。掌握这些工具可为你了解IDA的用户界面以及它显示的许多信息提供极大帮助。

[1] 请参阅http://www.cygwin.com/。

[2] 请参阅http://www.mingw.org/。

[3] 幻数是一些文件格式规范所要求的特殊标签值，它表示文件符合这种规范。有时，人们在选择幻数时加入了幽默的因素。例如，MS-DOS的可执行文件头中的MZ标签是MS-DOS原架构师Mark Zbikwsk姓名的首字母缩写。众所周知，Java的.class文件的幻数为十六进制数0xcafebabe，选择它作为幻数，仅仅是因为它是一个容易记忆的十六进制数字符串。

[4] 请参阅http://petools.org.ru/petools.shtml。

[5] 请参阅http://peid.info/。

[6] 请参阅http://securityresponse.symantec.com/security_response/writeup.jsp?docid=2003-112112-1102-99。

[7] 有关链接的更多信息，请参阅John R. Levine所著的《Linkers and Loaders》(San Francisco: Morgan Kaufmann, 2000)。

[8] 请参阅http://www.sourceware.org/binutils/docs/binutils/objdump.html#objdump/。

[9] 请参阅http://www.gnu.org/software/binutils/。

[10] 请参阅http://msdn.microsoft.com/en-us/library/c1h23y6c(VS.71).aspx。

[11] 有关名称改编的概述，请参考http://en.wikipedia.org/wiki/Name_mangling。

[12] 请参阅http://www.ragestorm.net/distorm/。

[13] 请参阅http://nasm.sourceforge.net/。

[14] 请参阅http://www.metasploit.com/。