标 题: LINUX的crackme的逆向小记(全文完)  
作 者: 采臣·宁  
时 间: 2005-02-20,14:11:56  
链 接: http://bbs.pediy.com/showthread.php?t=11315  
  
 **一个简单的linux crackme的逆向**  
  
前言  
  
    最不喜欢的就是写破解教程，酒后一时冲动，老夫卿发少年狂，许下将写一篇linux平台逆向的文章的诺言，作此文实非颇不得已。  
    在此申明:本文在技术上非常初级，并没有用到什么高深的技术，本人水平亦有限，如有差错，还请见谅！  
      
开始之前的准备  
  
    正如C语言教程从 hello world 开始，我们也由一个 crackme 说开去。本文的例子程序你可以到这来下载:  
http://www.crackmes.de/users/veneta/crackmes/linux\_crackme\_v2 。古人云“工欲善其事，必先利其器”，本文中所用到的工具及操作平台罗列如下:  
  操作平台: gentoo 2004.3 #  kernel 2.6.9  
  逆向工具:   
    反汇编 -- objdump （这个工具基本上每个LINUX上都有）、lida( http://lida.sourceforge.net/ ), 安装lida之前需要使用新立得来 安装libdisasm 0.16  
    调试器 -- gdb  
    十六进制编辑器 -- hexedit  
    文本编辑器 -- vim  
    压缩工具 -- upx  (http://upx.sourceforge.net)  
    计算器 -- gcalctool（gnome计算器）  
      
开始逆向之旅  
  
首先我们看看程序基本信息：  
    打开控制台，切换到程序所在目录。运行“ objdump -x cm2 ”，显示如下：

代码:

[ncc2008@localhost]$ objdump -x cm2

cm2:     file format elf32-i386

cm2

architecture: i386, flags 0x00000102:

EXEC\_P, D\_PAGED

start address 0x08048080

程序头：

    LOAD off    0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2\*\*12

         filesz 0x000005b8 memsz 0x000005b8 flags r-x

    LOAD off    0x000005b8 vaddr 0x080495b8 paddr 0x080495b8 align 2\*\*12

         filesz 0x0000002c memsz 0x0000002c flags rw-

Sections:

Idx Name          Size      VMA       LMA       File off  Algn

SYMBOL TABLE:

no symbols

    我们可以看到start address是0x08048080，但有一个问题是Sections下面却什么都没有。这不是一个正常的程序？  
      
    接下来，使用十六进制工具 hexedit 查看程序信息。运行命令：hexedit cm2 ,显示如下：

代码:

00000000   7F 45 4C 46  01 01 01 00  4C 69 6E 75  78 00 00 00  .ELF....Linux...

00000010   02 00 03 00  01 00 00 00  80 80 04 08  34 00 00 00  ............4...

00000020   00 00 00 00  00 00 00 00  34 00 20 00  02 00 00 00  ........4. .....

00000030   00 00 00 00  01 00 00 00  00 00 00 00  00 80 04 08  ................

00000040   00 80 04 08  B8 05 00 00  B8 05 00 00  05 00 00 00  ................

00000050   00 10 00 00  01 00 00 00  B8 05 00 00  B8 95 04 08  ................

00000060   B8 95 04 08  2C 00 00 00  2C 00 00 00  06 00 00 00  ....,...,.......

00000070   00 10 00 00  77 6C EA 93  7F 55 50 58  E4 05 0B 0A  ....wl...UPX....

00000080   31 ED 58 89  E1 8D 54 81  04 50 83 E4  F8 52 51 E8  1.X...T..P...RQ.

00000090   FE 01 00 00  F4 0A 00 24  49 6E 66 6F  3A 20 54 68  .......$Info: Th

000000A0   69 73 20 66  69 6C 65 20  69 73 20 70  61 63 6B 65  is file is packe

000000B0   64 20 77 69  74 68 20 74  68 65 20 55  50 58 20 65  d with the UPX e

000000C0   78 65 63 75  74 61 62 6C  65 20 70 61  63 6B 65 72  xecutable packer

000000D0   20 68 74 74  70 3A 2F 2F  75 70 78 2E  73 66 2E 6E   http://upx.sf.n

000000E0   65 74 20 24  0A 00 24 49  64 3A 20 55  50 58 20 31  et $..$Id: UPX 1

000000F0   2E 32 34 20  43 6F 70 79  72 69 67 68  74 20 28 43  .24 Copyright (C

00000100   29 20 31 39  39 36 2D 32  30 30 32 20  74 68 65 20  ) 1996-2002 the

00000110   55 50 58 20  54 65 61 6D  2E 20 41 6C  6C 20 52 69  UPX Team. All Ri

00000120   67 68 74 73  20 52 65 73  65 72 76 65  64 2E 20 24  ghts Reserved. $

    从上面的信息中可以看到程序被UPX压缩了，接下来请确信你系统中已有UPX，如果没有请到上面给出的程序链接中下载。  
    运行命令“ upx -d cm2 ”把程序解压缩，显示如下：

代码:

[ncc2008@localhost crack]$ upx -d cm2

                     Ultimate Packer for eXecutables

   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004

UPX 1.25         Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name

   --------------------   ------   -----------   -----------

     10584 <-      4870   46.01%    linux/386    cm2

Unpacked 1 file.

    接下来，我们再使用命令“ objdump -x cm2 ”来查看程序信息。如下：

代码:

[ncc2008@localhost crack]$ objdump -x cm2

cm2:     file format elf32-i386

cm2

architecture: i386, flags 0x00000112:

EXEC\_P, HAS\_SYMS, D\_PAGED

start address 0x080488b0

程序头：

    PHDR off    0x00000034 vaddr 0x08048034 paddr 0x08048034 align 2\*\*2

         filesz 0x000000c0 memsz 0x000000c0 flags r-x

  INTERP off    0x000000f4 vaddr 0x080480f4 paddr 0x080480f4 align 2\*\*0

         filesz 0x00000013 memsz 0x00000013 flags r--

    LOAD off    0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2\*\*12

         filesz 0x0000109c memsz 0x0000109c flags r-x

    LOAD off    0x00002000 vaddr 0x0804a000 paddr 0x0804a000 align 2\*\*12

         filesz 0x00000398 memsz 0x00000ff0 flags rw-

 DYNAMIC off    0x00002248 vaddr 0x0804a248 paddr 0x0804a248 align 2\*\*2

         filesz 0x000000e0 memsz 0x000000e0 flags rw-

    NOTE off    0x00000108 vaddr 0x08048108 paddr 0x08048108 align 2\*\*2

         filesz 0x00000020 memsz 0x00000020 flags r--

动态节：

  NEEDED      libgtk-1.2.so.0

  NEEDED      libgdk-1.2.so.0

  NEEDED      libglib-1.2.so.0

  NEEDED      libc.so.6

  INIT        0x8048758

  FINI        0x8049074

  HASH        0x8048128

  STRTAB      0x804841c

  SYMTAB      0x80481fc

  STRSZ       0x225

  SYMENT      0x10

  DEBUG       0x0

  PLTGOT      0x804a33c

  PLTRELSZ    0x98

  PLTREL      0x11

  JMPREL      0x80486c0

  REL         0x80486b8

  RELSZ       0x8

  RELENT      0x8

  VERNEED     0x8048688

  VERNEEDNUM  0x1

  VERSYM      0x8048642

版本引用：

  required from libc.so.6:

    0x0d696911 0x00 03 GLIBC\_2.1

    0x0d696910 0x00 02 GLIBC\_2.0

Sections:

Idx Name          Size      VMA       LMA       File off  Algn

  0 .interp       00000013  080480f4  080480f4  000000f4  2\*\*0

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  1 .note.ABI-tag 00000020  08048108  08048108  00000108  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  2 .hash         000000d4  08048128  08048128  00000128  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  3 .dynsym       00000220  080481fc  080481fc  000001fc  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  4 .dynstr       00000225  0804841c  0804841c  0000041c  2\*\*0

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  5 .gnu.version  00000044  08048642  08048642  00000642  2\*\*1

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  6 .gnu.version\_r 00000030  08048688  08048688  00000688  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  7 .rel.dyn      00000008  080486b8  080486b8  000006b8  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  8 .rel.plt      00000098  080486c0  080486c0  000006c0  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

  9 .init         00000017  08048758  08048758  00000758  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, CODE

 10 .plt          00000140  08048770  08048770  00000770  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, CODE

 11 .text         000007c4  080488b0  080488b0  000008b0  2\*\*4

                  CONTENTS, ALLOC, LOAD, READONLY, CODE

 12 .fini         0000001b  08049074  08049074  00001074  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, CODE

 13 .rodata       00000008  08049090  08049090  00001090  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

 14 .eh\_frame     00000004  08049098  08049098  00001098  2\*\*2

                  CONTENTS, ALLOC, LOAD, READONLY, DATA

 15 .data         00000248  0804a000  0804a000  00002000  2\*\*2

                  CONTENTS, ALLOC, LOAD, DATA

 16 .dynamic      000000e0  0804a248  0804a248  00002248  2\*\*2

                  CONTENTS, ALLOC, LOAD, DATA

 17 .ctors        00000008  0804a328  0804a328  00002328  2\*\*2

                  CONTENTS, ALLOC, LOAD, DATA

 18 .dtors        00000008  0804a330  0804a330  00002330  2\*\*2

                  CONTENTS, ALLOC, LOAD, DATA

 19 .jcr          00000004  0804a338  0804a338  00002338  2\*\*2

                  CONTENTS, ALLOC, LOAD, DATA

 20 .got          0000005c  0804a33c  0804a33c  0000233c  2\*\*2

                  CONTENTS, ALLOC, LOAD, DATA

 21 .bss          00000c58  0804a398  0804a398  00002398  2\*\*2

                  ALLOC

 22 .comment      00000119  00000000  00000000  00002398  2\*\*0

                  CONTENTS, READONLY

SYMBOL TABLE:

no symbols

    从上面的信息中可以看出，程序使用gtk编写，程序入口start address＝0x080488b0。  
    我们再进一步的查看程序信息，运行命令“objdump -T cm2 >iat.txt”，下面的信息将会保存到iat.txt文件中。

代码:

[ncc2008@localhost crack]$ objdump -T cm2

cm2:     file format elf32-i386

DYNAMIC SYMBOL TABLE:

00000000      DF \*UND\*  0000013d              gtk\_widget\_show

0804a248 g    DO \*ABS\*  00000000  Base        \_DYNAMIC

08048780      DF \*UND\*  000000e4              gtk\_widget\_show\_all

00000000      DF \*UND\*  00000027              gtk\_dialog\_new

08048790      DF \*UND\*  000001db              gtk\_container\_add

08048758 g    DF .init  00000000  Base        \_init

080487a0      DF \*UND\*  000000d1              gtk\_window\_set\_default\_size

080487b0      DF \*UND\*  00000056              gtk\_label\_new

080487c0      DF \*UND\*  00000085              gtk\_window\_new

080487d0      DF \*UND\*  00000213              gtk\_widget\_set\_uposition

080487e0      DF \*UND\*  0000010d              gtk\_window\_set\_title

080487f0      DF \*UND\*  0000010c              gtk\_entry\_get\_text

08048800      DF \*UND\*  00000242              gtk\_box\_pack\_start

08048810      DF \*UND\*  00000067              gtk\_init

08048820      DF \*UND\*  0000010d              g\_print

08048830      DF \*UND\*  0000007c              gtk\_main\_quit

08048840      DF \*UND\*  000001d3              gtk\_main

0804a398 g    D  \*ABS\*  00000000  Base        \_\_bss\_start

08048850      DF \*UND\*  000000e0  GLIBC\_2.0   \_\_libc\_start\_main

08049074 g    DF .fini  00000000  Base        \_fini

08048860      DF \*UND\*  000000c7  GLIBC\_2.0   exit

0804a398 g    D  \*ABS\*  00000000  Base        \_edata

0804a33c g    DO \*ABS\*  00000000  Base        \_GLOBAL\_OFFSET\_TABLE\_

0804aff0 g    D  \*ABS\*  00000000  Base        \_end

00000000      DF \*UND\*  00000028  GLIBC\_2.1   fopen

08049094 g    DO .rodata        00000004  Base        \_IO\_stdin\_used

00000000      DF \*UND\*  0000012e  GLIBC\_2.0   fwrite

08048870      DF \*UND\*  00000027              gtk\_entry\_new

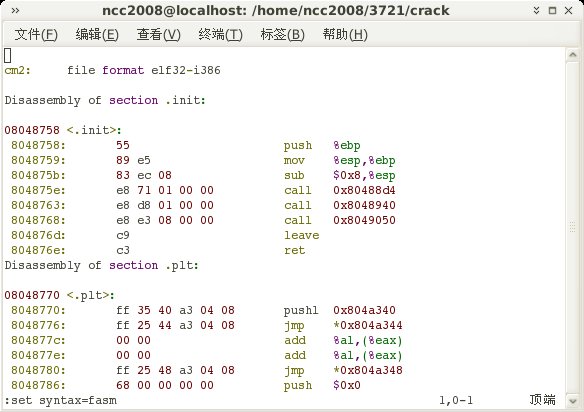
00000000  w   D  \*UND\*  00000000              \_Jv\_RegisterClasses

08048880      DF \*UND\*  00000157              gtk\_signal\_connect

08048890      DF \*UND\*  0000006d              gtk\_button\_new\_with\_label

080488a0      DF \*UND\*  00000044              gtk\_vbox\_new

00000000  w   D  \*UND\*  00000000              \_\_gmon\_start\_\_

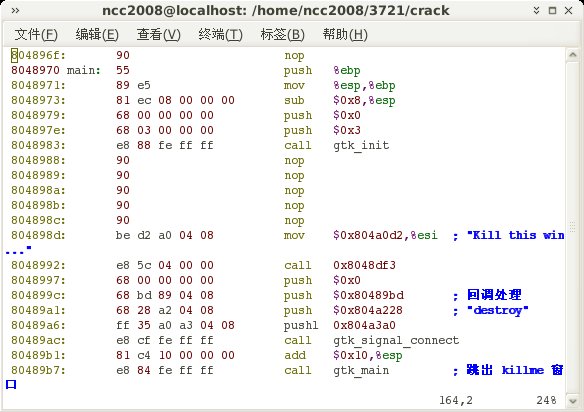
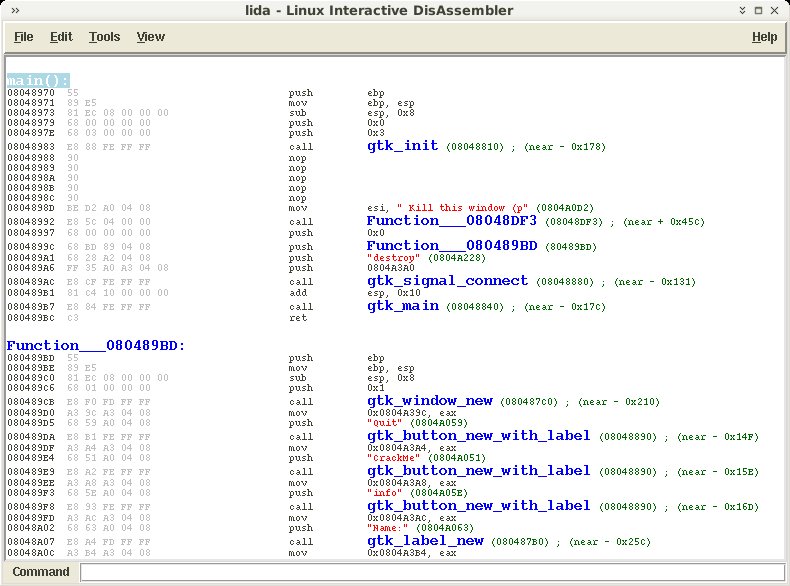
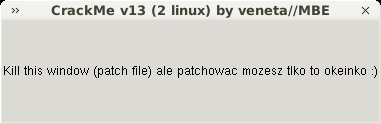
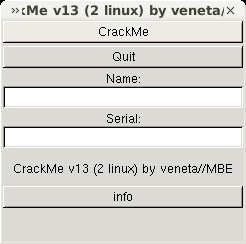
    我们再一次运行命令“objdump -d cm2 >disasm.txt”，对程序进行反汇编，并把结果保存到disasm.txt中。  
接下来，用VI打开disasm.txt文件。显示如下：  
  
      
相对WINDOW平台的反汇编器，objdump 的反汇编结果在没有调试符号的情况下，不显示API的名字。下一步API调用手工的加上。  
我们打开iat.txt文件，把disasm.txt文件中的地址全部转换成函数名，例如：

代码:

 8048979:       68 00 00 00 00          push   $0x0

 804897e:       68 03 00 00 00          push   $0x3

 8048983:       e8 88 fe ff ff          call   0x8048810

转换成  
  
  
这样将好看多了。这一点你可以写个脚本自动完成，而不用手工一个一个的替换。或者你也可以使用更强大的反汇编工器lida来自动显示API调用名。为了方便初学者，下面使用LIDA来作为反汇编工具。用lida 载入程序，如下图：  
  
    好了，前期的工作都做了。下面让我们来运行一下程序，看看有没有什么值得参考的。运行后，出现一个窗口。如下：  
   
    如图，提示我们的CRACK任务之一就是NOP掉这个窗口。点击关闭按钮，出现第二个窗口，如下图  
  
    这是注册的主窗口。  
    对照LIDA中的反汇编代码加上GTK的一些基本知识，我们可以知道程序运行过程是这样的：main->跳出killme窗口->注册destroy事件处理函数->在关闭 killme 窗口时将引发destroy事件->运行回调处理函数0x80489bd->跳出crackme窗口。  
    我们CRACK的任务就是nop掉killme窗口，以及找出注册码。  
    注：GTK的相关文档你可以到这来获取 http://www.gtk.org/tutorial/  
  
  
第一个任务：去除killme窗口。  
    用lida反汇编cm2，点击菜单 view -> Functions 找到main函数，点击进入，如下所示：

代码:

main():

08048970  55                                   push      ebp

08048971  89 E5                                mov      ebp, esp

08048973  81 EC 08 00 00 00                    sub      esp, 0x8

08048979  68 00 00 00 00                       push      0x0

0804897E  68 03 00 00 00                       push      0x3

08048983  E8 88 FE FF FF                       call      gtk\_init (08048810) ; (near - 0x178)

08048988  90                                   nop  -\

08048989  90                                   nop    \

0804898A  90                                   nop     |->怎么会有nop指令出现？

0804898B  90                                   nop    /   我们在这加入call Function\_\_\_080489BD

0804898C  90                                   nop  -/    直接显示crackme窗口，跳过killme窗口的调用。因为在

0804898D  BE D2 A0 04 08                       mov      esi, " Kill this window (p" (0804A0D2)

08048992  E8 5C 04 00 00                       call      Function\_\_\_08048DF3 (08048DF3) ; 建立killme窗口

08048997  68 00 00 00 00                       push      0x0

0804899C  68 BD 89 04 08                       push      Function\_\_\_080489BD (80489BD)  ; 回调函数，显示crackme窗口

080489A1  68 28 A2 04 08                       push      "destroy" (0804A228)

080489A6  FF 35 A0 A3 04 08                    push      0804A3A0

080489AC  E8 CF FE FF FF                       call      gtk\_signal\_connect (08048880) ; 建立destroy事件回调函数0x080489BD

080489B1  81 C4 10 00 00 00                    add      esp, 0x10

080489B7  E8 84 FE FF FF                       call      gtk\_main (08048840) ; 显示killme窗口

080489BC  C3                                   ret

    我们跟入Function\_\_\_08048DF3函数，如下：

代码:

Function\_\_\_08048DF3:

08048DF3  55                                   push      ebp  ; xref ( 08048992 08048C9A 08048CA6 08048DAC )

08048DF4  89 E5                                mov      ebp, esp

08048DF6  81 EC 08 00 00 00                    sub      esp, 0x8

08048DFC  68 01 00 00 00                       push      0x1

08048E01  E8 BA F9 FF FF                       call      gtk\_window\_new (080487C0) ; (near - 0x646)

08048E06  A3 A0 A3 04 08                       mov      0x0804A3A0, eax

08048E0B  68 2C A0 04 08                       push      "CrackMe v13 (2 linux" (0804A02C)

08048E10  FF 35 A0 A3 04 08                    push      0804A3A0

08048E16  E8 C5 F9 FF FF                       call      gtk\_window\_set\_title (080487E0) ; (near - 0x63B)

08048E1B  56                                   push      esi

08048E1C  E8 8F F9 FF FF                       call      gtk\_label\_new (080487B0) ; (near - 0x671)

08048E21  50                                   push      eax

08048E22  FF 35 A0 A3 04 08                    push      0804A3A0

08048E28  E8 63 F9 FF FF                       call      gtk\_container\_add (08048790) ; (near - 0x69D)

08048E2D  68 64 00 00 00                       push      0x64

08048E32  68 2C 01 00 00                       push      0x12C

08048E37  FF 35 A0 A3 04 08                    push      0804A3A0

08048E3D  E8 5E F9 FF FF                       call      gtk\_window\_set\_default\_size (080487A0) ; (near - 0x6A2)

08048E42  FF 35 A0 A3 04 08                    push      0804A3A0

08048E48  E8 33 F9 FF FF                       call      gtk\_widget\_show\_all (08048780) ; (near - 0x6CD)

08048E4D  C9                                   leave

08048E4E  C3                                   ret

    上面这个函数建立killme窗口。  
    为了去除这个killme窗口。我们可以在

代码:

08048988  90                                   nop  -\

08048989  90                                   nop    \

0804898A  90                                   nop     |->怎么会有nop指令出现？

0804898B  90                                   nop    /   我们在这加入call Function\_\_\_080489BD

0804898C  90                                   nop  -/    直接显示crackme窗口，跳过killme窗口的调用。

0804898D  BE D2 A0 04 08                       mov      esi, " Kill this window (p" (0804A0D2)

08048992  E8 5C 04 00 00                       call      Function\_\_\_08048DF3 (08048DF3)

    上面的nop正好5个，我怀疑作者写crackme在这有一个对crackme函数的调用，生成程序后，他手工把这个代码NOP掉了，他是故意留出空间来做提示。因为在crackme 窗口函数中有完整的窗口初始化和退出处理，如下：

代码:

08048C22  68 3D A2 04 08                       push      "clicked" (0804A23D)

08048C27  FF 35 AC A3 04 08                    push      0804A3AC

08048C2D  E8 4E FC FF FF                       call      gtk\_signal\_connect (08048880) ; (near - 0x3B2)

08048C32  81 C4 10 00 00 00                    add      esp, 0x10

08048C38  E8 03 FC FF FF                       call      gtk\_main (08048840) ; (near - 0x3FD)

08048C3D  68 2C A0 04 08                       push      "CrackMe v13 (2 linux" (0804A02C)

08048C42  E8 D9 FB FF FF                       call      g\_print (08048820) ; (near - 0x427)

08048C47  81 C4 04 00 00 00                    add      esp, 0x4

08048C4D  68 00 00 00 00                       push      0x0

08048C52  E8 09 FC FF FF                       call      exit (08048860) ; 当点击crackme窗口上的quit按钮后程序直接退出

Function\_\_\_08048C57:

08048C57  55                                   push      ebp

08048C58  89 E5                                mov      ebp, esp

08048C5A  81 EC 08 00 00 00                    sub      esp, 0x8

08048C60  E8 17 00 00 00                       call      Function\_\_\_08048C7C (08048C7C) ; (near + 0x17)

08048C65  B8 00 00 00 00                       mov      eax, 0x0

08048C6A  C9                                   leave

08048C6B  C3                                   ret

    从上面的代码可以看到，如果先调用crackme 窗口的函数，那么 killme 窗口就不会出现了。  
    好的，下面我们就改08048988处的指令为 call    080489BD ，用计算器算一下:80489BD-08048988-5=00000030，得到代码为 e8 30 00 00 00 。嘿嘿！  
    好的，用hexedit 打开 cm2 程序。看看前面objdump -x cm2列出的基地址为0x8048000，所以对应的file offset应该是0x8048988-0x8048000=0x988，我们按 return 键，打开 go to 窗口，输入 0x988 ，定位到08048988处，改代码为e830000000。改完后，“ ./cm2 ”运行程序，OK。出现 crackme 窗口了。  
  
第二个任务，找出正确的注册码。  
为保持本文一个苗条的身材及加大本文所包含的信息量，我就不讲怎么根据按钮名找对应的点击事件处理函数了，讲一种简单省事的方法，找注册码最省事的方法当然是动态跟踪啦。搞不好还可以看到明码哟。费话不多说，运行命令： gdb cm2  。提示没有符号文件，不用管它。  
下断点:

代码:

br gtk\_entry\_get\_text 注（类似于WIN平台下的 getdlgitemtexta ）

再键入命令 “ run ”运行程序。出现了界面。  
输入名字和注册码，我这用的是:  
名字：ncc2008  
注册码：78787878  
按crackme按钮，中断到gdb中。

代码:

Breakpoint 1, 0x40096eb0 in gtk\_entry\_get\_text () from /usr/lib/libgtk-1.2.so.0

让我们看看是哪个CALL调用的，下命令"info frame"，显示如下：

代码:

(gdb) info frame

Stack level 0, frame at 0xbfffe250:

 eip = 0x40096eb0 in gtk\_entry\_get\_text; saved eip 0x8048cc1  ；从这CALL来的

 called by frame at 0xbfffe254

 Arglist at 0xbfffe248, args:

 Locals at 0xbfffe248, Previous frame's sp is 0xbfffe250

 Saved registers:

  eip at 0xbfffe24c

呵呵，清除全部断点，下命令“d”。在0x8048cc1下一个断点：

代码:

br \*0x8048cc1

，然后键“C”继续运行程序。  
程序中断到我们所下的断点0x8048cc1处。让我们看看汇编代码：  
下命令：x /10i $eip  
显示如下：

代码:

(gdb) x /10i $eip

0x8048cc1:      mov    %eax,%edx

0x8048cc3:      call   0x8048fc5

0x8048cc8:      cmp    $0x20,%eax

0x8048ccd:      jne    0x8048ca1

0x8048ccf:      mov    %edx,0x804a3e8

0x8048cd5:      call   0x8048f50

0x8048cda:      pushl  0x804a3c0

0x8048ce0:      call   0x80487f0

0x8048ce5:      mov    %eax,%edx

0x8048ce7:      call   0x8048fc5

看不懂？我们再看lida 中的 0x8048cc1处的代码。

代码:

Function\_\_\_08048CAD:

08048CAD  55                                   push      ebp

08048CAE  89 E5                                mov      ebp, esp

08048CB0  81 EC 08 00 00 00                    sub      esp, 0x8

08048CB6  FF 35 C4 A3 04 08                    push      0804A3C4

08048CBC  E8 2F FB FF FF                       call      gtk\_entry\_get\_text (080487F0) ; (near - 0x4D1)

08048CC1  89 C2                                mov      edx, eax

08048CC3  E8 FD 02 00 00                       call      Function\_\_\_08048FC5 (08048FC5) ; 注册码长

08048CC8  3D 20 00 00 00                       cmp      eax, 0x20

08048CCD  75 D2                                jnz      Label\_08048CA1 (08048CA1) ; 注册码必需是32个字符

08048CCF  89 15 E8 A3 04 08                    mov      0804A3E8, edx

08048CD5  E8 76 02 00 00                       call      Function\_\_\_08048F50 (08048F50) ; a2i，字符串到 hex转换

08048CDA  FF 35 C0 A3 04 08                    push      0804A3C0

08048CE0  E8 0B FB FF FF                       call      gtk\_entry\_get\_text (080487F0) ; (near - 0x4F5)

08048CE5  89 C2                                mov      edx, eax

08048CE7  E8 D9 02 00 00                       call      Function\_\_\_08048FC5 (08048FC5) ; (near + 0x2D9)

08048CEC  BE F0 A7 04 08                       mov      esi, 0x804A7F0

08048CF1  50                                   push      eax

08048CF2  3D 03 00 00 00                       cmp      eax, 0x3

08048CF7  72 A8                                jc      Label\_08048CA1 (08048CA1) ; 用户名必需大于3个字符

08048CF9  89 C1                                mov      ecx, eax

08048CFB  8A 02                                mov      al, [edx]  ; xref ( 08048D01 )

08048CFD  88 06                                mov      [esi], al

08048CFF  42                                   inc      edx

08048D00  46                                   inc      esi

08048D01  E2 F8                                loop      08048CFB ; (near  0xF8)

08048D03  C7 05 C8 A3 04 08 10 A2 04 08        mov      0804A3C8, "MBECrew" (0804A210)

08048D0D  C7 05 CC A3 04 08 07 00 00 00        mov      0804A3CC, 0x7

08048D17  E8 33 01 00 00                       call      Function\_\_\_08048E4F (08048E4F) ; (near + 0x133)

08048D1C  C7 05 D0 A3 04 08 F0 A3 04 08        mov      0804A3D0, 0x804A3F0

08048D26  C7 05 D4 A3 04 08 10 00 00 00        mov      0804A3D4, 0x10

08048D30  E8 87 01 00 00                       call      Function\_\_\_08048EBC (08048EBC) ; 注册码每字节与F0相异或

08048D35  C7 05 C8 A3 04 08 10 A2 04 08        mov      0804A3C8, "MBECrew" (0804A210)

08048D3F  C7 05 CC A3 04 08 07 00 00 00        mov      0804A3CC, 0x7

08048D49  E8 01 01 00 00                       call      Function\_\_\_08048E4F (08048E4F) ; (near + 0x101)

08048D4E  58                                   pop      eax

08048D4F  C7 05 D0 A3 04 08 F0 A7 04 08        mov      0804A3D0, 0x804A7F0

08048D59  A3 D4 A3 04 08                       mov      0x0804A3D4, eax

08048D5E  E8 59 01 00 00                       call      Function\_\_\_08048EBC (08048EBC) ; (near + 0x159)

08048D63  BE F0 A3 04 08                       mov      esi, 0x804A3F0

08048D68  BA F0 A7 04 08                       mov      edx, 0x804A7F0

08048D6D  B9 10 00 00 00                       mov      ecx, 0x10

Label\_08048D72:

08048D72  8A 02                                mov      al, [edx]  ; xref ( 08048D7C 08048DB8 )

08048D74  3C 00                                cmp      al, 0x0

08048D76  74 3B                                jz      Label\_08048DB3 (08048DB3) ; (near + 0x3B)

08048D78  04 0A                                add      al, 0xA

08048D7A  30 06                                xor      [esi], al

08048D7C  E2 F4                                loop      Label\_08048D72 (08048D72) ; (near  0xF4)

08048D7E  BA 18 A2 04 08                       mov      edx, 0x804A218

08048D83  BE F0 A3 04 08                       mov      esi, 0x804A3F0

08048D88  B9 04 00 00 00                       mov      ecx, 0x4

08048D8D  8B 02                                mov      eax, [edx]  ; xref ( 08048DA5 )

08048D8F  33 06                                xor      eax, [esi]  ; 下断点在这

08048D91  85 C0                                test      eax, eax

08048D93  0F 85 08 FF FF FF                    jnz      Label\_08048CA1 (08048CA1) ; 跳就死定了

08048D99  81 C2 04 00 00 00                    add      edx, 0x4

08048D9F  81 C6 04 00 00 00                    add      esi, 0x4

08048DA5  E2 E6                                loop      08048D8D ; (near  0xE6)

08048DA7  BE 8F A0 04 08                       mov      esi, " cracked ohh yeah !!" (0804A08F)

08048DAC  E8 42 00 00 00                       call      Function\_\_\_08048DF3 (08048DF3) ; (near + 0x42)

08048DB1  C9                                   leave

08048DB2  C3                                   ret

Label\_08048DB3:

08048DB3  BA F0 A7 04 08                       mov      edx, 0x804A7F0  ; xref ( 08048D76 )

08048DB8  EB B8                                jmp      Label\_08048D72 (08048D72) ; (near  0xB8)

接下来，我们就一步一步跟吧！整个过程单调无趣，不多说了。注册码必需是32位，我使用“12345678abcdef0123456789abcdef0”  
跟踪其流程：  
转换注册码“12345678abcdef0123456789abcdef0”为：

代码:

0x804a3f0:      0x12    0x34    0x56    0x78    0x9a    0xbc    0xde    0xf0

0x804a3f8:      0x12    0x34    0x56    0x78    0x9a    0xbc    0xde    0xf0

然后每一字节与"0xf0"相异或，生成：

代码:

0x804a3f0:      0xe2    0xc4    0xa6    0x88    0x6a    0x4c    0x2e    0x00

0x804a3f8:      0xe2    0xc4    0xa6    0x88    0x6a    0x4c    0x2e    0x00

我们在08048D8F下个断点，中断后可以看到正确的值，edx中的值为：

代码:

(gdb) x /20xb $edx

0x804a218:      0xc4    0x84    0x0b    0xb5    0xa1    0x02    0x0d    0x97

0x804a220:      0x5c    0xb2    0xe1    0xc9    0xa4    0x91    0x52    0xcc

0x804a218处的值与0x804a3f0处的值相比较，相等就注册成功。结合前面注册码的变换，可以把上面的每字节与F0异或就是正确的注册码了，掏出计算器，计算如下：

代码:

3474fb4551f2fd67ac4211395461a23c

注册码与名字无关，名字可随便写，在程序中输入上面的注册码，成功！  
  
补丁和注册机代码就不发作者了，古得白  
全文完