Red Team Module 0:
Crash Course X86 Binaries

Introduction:

When you hear the word "binary" the first thing that might jump into your head are the numbers 1 and 0. Fundamentally, computers can understand two states: HIGH and LOW. When we write a program in a language like c, high level instructions are transformed from something that can be easily read by humans into something that can be executed by computers.

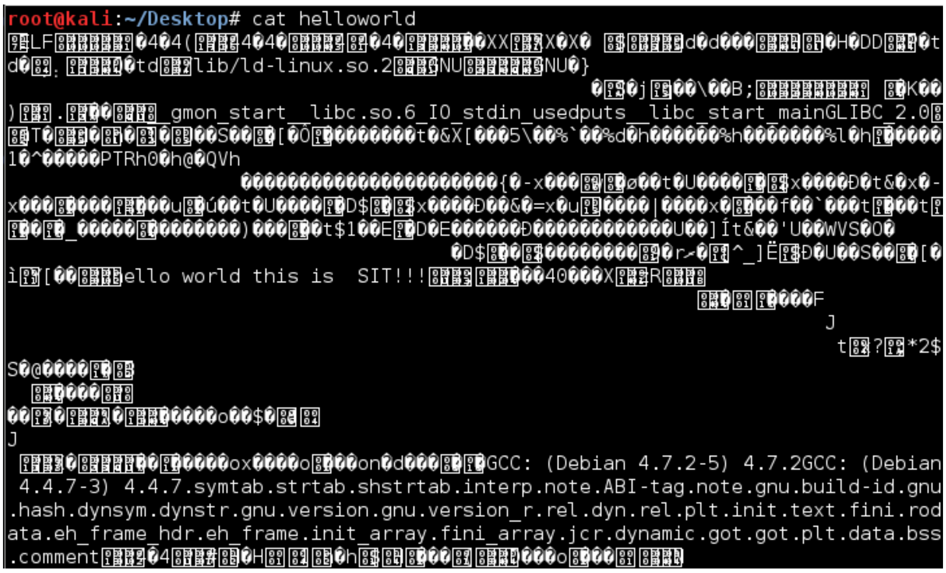
Background:

Modern operating systems use a type of binary called an executable to "execute" instructions:

1. Linux – ELF executable
2. Mac – MachO executable
3. Windows – EXE executable

For now we are only going to take a look at ELF executables We can easily reverse engineer these files from the kali virtualmachines installed in BLUE-MOD-0.

Lets start by taking a look at a compiled program called "helloworld.c" by printing its contents to standard out. At first glance what we get appears to be rather cryptic...



```
root@kali:~/Desktop# cat helloworld
▯LF▯▯▯▯▯▯▯�4�4(▯▯▯4�4�▯▯▯▯▯▯▯�4�▯▯▯▯▯▯�XX▯▯X�X� ▯▯▯▯▯▯d�d���▯▯▯▯▯▯�H�DD▯▯▯�t
d�▯. ▯▯▯▯▯�td▯▯lib/ld-linux.so.2▯▯▯GNU▯▯▯▯▯▯GNU�}
                                                 �▯�j▯▯��\��B;▯▯▯▯▯▯▯▯▯  ▯▯K��
)▯▯.▯▯�▯▯▯_gmon_start__libc.so.6_IO_stdin_usedputs__libc_start_mainGLIBC_2.0▯
▯T�▯▯▯�▯▯�▯▯�▯▯��S��▯[�Ō▯��������t�&X[���5\��%`��%d�h������%h�������%l�h▯����
1�^�����PTRh0�h@�QVh
                         ����������������������{�-x���▯▯▯▯ø��t�U����▯▯▯▯x����Ð�t&�x�-
x���▯����▯▯��u▯▯ú��t�U����▯▯D$▯▯▯▯x����Ð��&�=x�u▯▯����|����x�▯▯���f��`���t▯▯���t▯
▯▯��▯▯_�����▯▯��������)���▯▯�t$1��E▯▯D�E������Ð�������������U��]Ít&��'U��WVS�O�
                                                 �D$▯▯��▯▯���������▯▯�r~�▯▯^_]Ë▯▯Ð�U��S��▯▯[�
ì▯▯[��▯▯▯ello world this is  SIT!!!▯▯▯▯▯▯▯▯��40���X▯▯▯R▯▯▯▯
                                                                              ▯▯�▯▯▯▯����F
                                                                                       J
                                                                                        t▯▯?▯▯*2$
S�@����▯▯▯▯
  ▯▯�����▯▯
��▯▯�▯▯▯▯�▯▯▯▯����o��$�▯▯ ▯▯
J
 ▯▯▯▯�▯▯▯▯▯▯▯��▯▯�����ox����o▯▯���on�d���▯▯▯▯GCC: (Debian 4.7.2-5) 4.7.2GCC: (Debian
 4.4.7-3) 4.4.7.symtab.strtab.shstrtab.interp.note.ABI-tag.note.gnu.build-id.gnu
.hash.dynsym.dynstr.gnu.version.gnu.version_r.rel.dyn.rel.plt.init.text.fini.rod
ata.eh_frame_hdr.eh_frame.init_array.fini_array.jcr.dynamic.got.got.plt.data.bss
.comment▯▯▯▯�4▯▯▯▯▯▯�H▯▯ ▯▯ ▯▯�h▯▯▯▯▯▯���▯▯ ▯▯▯▯���o▯▯���▯▯ ▯▯▯▯
```

This is the *ascii* representation of the bytes that compose instructions in the file. As you may imagine it doesn't make a lot of sense to reverse engineer binaries like this. Printing to our terminal's standard out/using a regular text editor can cause our terminal emulator to start behaving strange and even make sound. There is still some useful information we can extract from this picture such as some of the strings like "Hello world this is SIT" that compose the file, but this is probably done better with the linux command: *strings*.

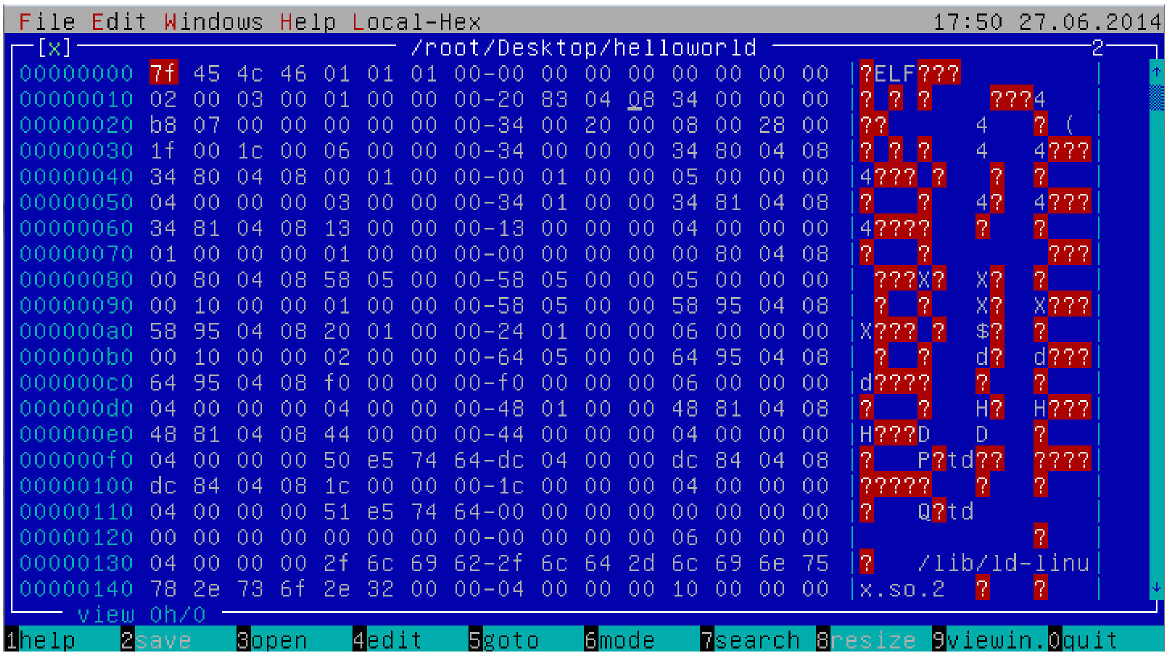
To make our analysis better we can look at the binary "helloworld" from inside a hex editor. In this case we will use the editor hte. This program allows us to not only look at the binary in hex, but also to edit the program and make changes to instructions.

hte does not come packaged with Kali. We can however retrieve it from the Debian repositories using the apt package manager. The following command should retrieve the binary:

> *sudo apt-get install ht*

Surprisingly this package does not have the same name as its binary. You need to type hte instead of ht to start it from a terminal.

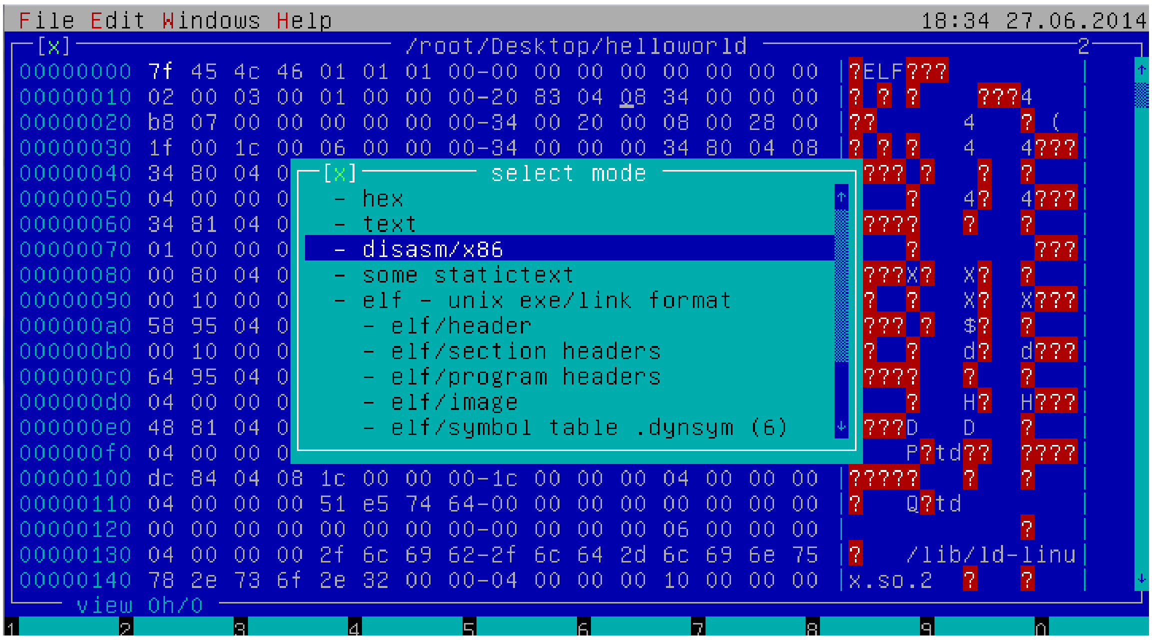
Hte is a terminal based application. However, it doesn't always run well in gnome's terminal emulator, I've had mixed results. If the formatting is giving you problems I recommend switching ttys and then logging into a blank shell. This can be done in gnome by using *ctrl+alt+F1*. Lets go ahead and open up the helloworld binary with *hte helloworld*.



```
 File Edit Windows Help Local-Hex                               17:50 27.06.2014
┌[x]───────────────── /root/Desktop/helloworld ─────────────────────────2┐
 00000000 7f 45 4c 46 01 01 01 00-00 00 00 00 00 00 00 00 |?ELF???          |
 00000010 02 00 03 00 01 00 00 00-20 83 04 08 34 00 00 00 |? ? ?     ???4   |
 00000020 b8 07 00 00 00 00 00 00-34 00 20 00 08 00 28 00 |??      4    ? ( |
 00000030 1f 00 1c 00 06 00 00 00-34 00 00 00 34 80 04 08 |? ? ?   4    4???|
 00000040 34 80 04 08 00 01 00 00-00 01 00 00 05 00 00 00 |4??? ?      ?  ? |
 00000050 04 00 00 00 03 00 00 00-34 01 00 00 34 81 04 08 |?    ?  4?   4???|
 00000060 34 81 04 08 13 00 00 00-13 00 00 00 04 00 00 00 |4????     ?    ? |
 00000070 01 00 00 00 01 00 00 00-00 00 00 00 00 80 04 08 |?    ?        ???|
 00000080 00 80 04 08 58 05 00 00-58 05 00 00 05 00 00 00 | ???X?   X?   ?  |
 00000090 00 10 00 00 01 00 00 00-58 05 00 00 58 95 04 08 | ?   ?   X?   X???|
 000000a0 58 95 04 08 20 01 00 00-24 01 00 00 06 00 00 00 |X??? ?   $?   ?  |
 000000b0 00 10 00 00 02 00 00 00-64 05 00 00 64 95 04 08 | ?   ?   d?   d???|
 000000c0 64 95 04 08 f0 00 00 00-f0 00 00 00 06 00 00 00 |d????     ?    ? |
 000000d0 04 00 00 00 04 00 00 00-48 01 00 00 48 81 04 08 |?    ?   H?   H???|
 000000e0 48 81 04 08 44 00 00 00-44 00 00 00 04 00 00 00 |H???D     D    ? |
 000000f0 04 00 00 00 50 e5 74 64-dc 04 00 00 dc 84 04 08 |?    P?td??  ????|
 00000100 dc 84 04 08 1c 00 00 00-1c 00 00 00 04 00 00 00 |?????     ?    ? |
 00000110 04 00 00 00 51 e5 74 64-00 00 00 00 00 00 00 00 |?    Q?td         |
 00000120 00 00 00 00 00 00 00 00-00 00 00 00 06 00 00 00 |               ? |
 00000130 04 00 00 00 2f 6c 69 62-2f 6c 64 2d 6c 69 6e 75 |?    /lib/ld-linu|
 00000140 78 2e 73 6f 2e 32 00 00-04 00 00 00 10 00 00 00 |x.so.2    ?    ? |
└──── view 0h/0 ─────────────────────────────────────────────────────────┘
1help 2save 3open 4edit 5goto 6mode 7search 8resize 9viewin.0quit
```

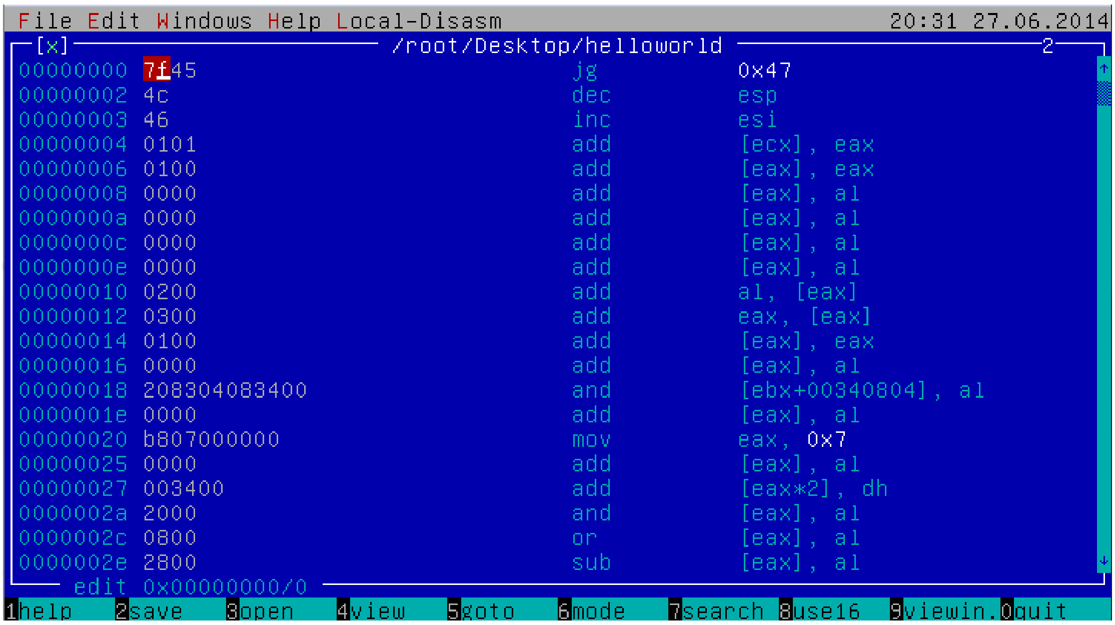
This is better, but staring at a wall of hexadecimal numbers is still rather painful to look at. The important concept to gather from this is that by ordering hexadecimal numbers in the right sequence we get opcodes. These are instructions that tell our CPU what to do.

Fortunately for us, hte can also act as a disassembler. The purpose of a disassembler is to translate op-codes back into assembly language. We can access hte's disassembly function using the space-bar.



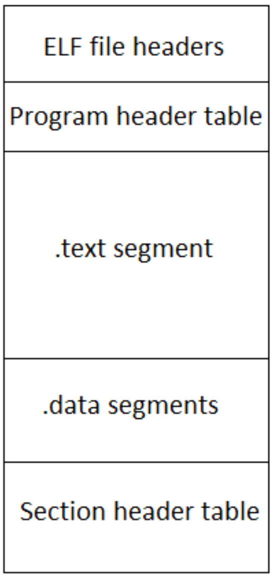
```
 File Edit Windows Help                                         18:34 27.06.2014
┌[x]───────────────── /root/Desktop/helloworld ─────────────────────────2┐
 00000000 7f 45 4c 46 01 01 01 00-00 00 00 00 00 00 00 00 |?ELF???          |
 00000010 02 00 03 00 01 00 00 00-20 83 04 08 34 00 00 00 |? ? ?     ???4   |
 00000020 b8 07 00 00 00 00 00 00-34 00 20 00 08 00 28 00 |??      4    ? ( |
 00000030 1f 00 1c 00 06 00 00 00-34 00 00 00 34 80 04 08 |? ? ?   4    4???|
 00000040 34 80 04 0┌[x]──────── select mode ─────────────┐??? ?      ?  ? |
 00000050 04 00 00 0│ - hex                                │    ?  4?   4???|
 00000060 34 81 04 0│ - text                               │????     ?    ? |
 00000070 01 00 00 0│ - disasm/x86                         │    ?        ???|
 00000080 00 80 04 0│ - some statictext                    │???X?   X?   ?  |
 00000090 00 10 00 0│ - elf - unix exe/link format         │?   ?   X?   X???|
 000000a0 58 95 04 0│   - elf/header                       │??? ?   $?   ?  |
 000000b0 00 10 00 0│   - elf/section headers              │?   ?   d?   d???|
 000000c0 64 95 04 0│   - elf/program headers              │????     ?    ? |
 000000d0 04 00 00 0│   - elf/image                        │    ?   H?   H???|
 000000e0 48 81 04 0│   - elf/symbol table .dynsym (6)     │???D     D    ? |
 000000f0 04 00 00 0└──────────────────────────────────────┘    P?td??  ????|
 00000100 dc 84 04 08 1c 00 00 00-1c 00 00 00 04 00 00 00 |?????     ?    ? |
 00000110 04 00 00 00 51 e5 74 64-00 00 00 00 00 00 00 00 |?    Q?td         |
 00000120 00 00 00 00 00 00 00 00-00 00 00 00 06 00 00 00 |               ? |
 00000130 04 00 00 00 2f 6c 69 62-2f 6c 64 2d 6c 69 6e 75 |?    /lib/ld-linu|
 00000140 78 2e 73 6f 2e 32 00 00-04 00 00 00 10 00 00 00 |x.so.2    ?    ? |
└──── view 0h/0 ─────────────────────────────────────────────────────────┘
1        2        3        4        5        6        7        8        9        0
```

Excellent! Now we have a direct translation from Hex to assembly. Unfortunately the picture is rather incomplete. In fact some translations don't make sense...



```
 File Edit Windows Help Local-Disasm                                     20:31 27.06.2014
┌[x]──────────────────────── /root/Desktop/helloworld ─────────────────────────2─┐
 00000000 7f45                          jg          0x47
 00000002 4c                            dec         esp
 00000003 46                            inc         esi
 00000004 0101                          add         [ecx], eax
 00000006 0100                          add         [eax], eax
 00000008 0000                          add         [eax], al
 0000000a 0000                          add         [eax], al
 0000000c 0000                          add         [eax], al
 0000000e 0000                          add         [eax], al
 00000010 0200                          add         al, [eax]
 00000012 0300                          add         eax, [eax]
 00000014 0100                          add         [eax], eax
 00000016 0000                          add         [eax], al
 00000018 208304083400                  and         [ebx+00340804], al
 0000001e 0000                          add         [eax], al
 00000020 b807000000                    mov         eax, 0x7
 00000025 0000                          add         [eax], al
 00000027 003400                        add         [eax*2], dh
 0000002a 2000                          and         [eax], al
 0000002c 0800                          or          [eax], al
 0000002e 2800                          sub         [eax], al
└──── edit 0x00000000/0 ─────────────────────────────────────────────────────────┘
1help    2save    3open    4view    5goto    6mode    7search 8use16   9viewin.0quit
```

As a brief overview the left most column represents the first byte of an op-code's place in the file. The second column on the left are the instruction's op-codes. The right hand side represents the actual assembly instructions.

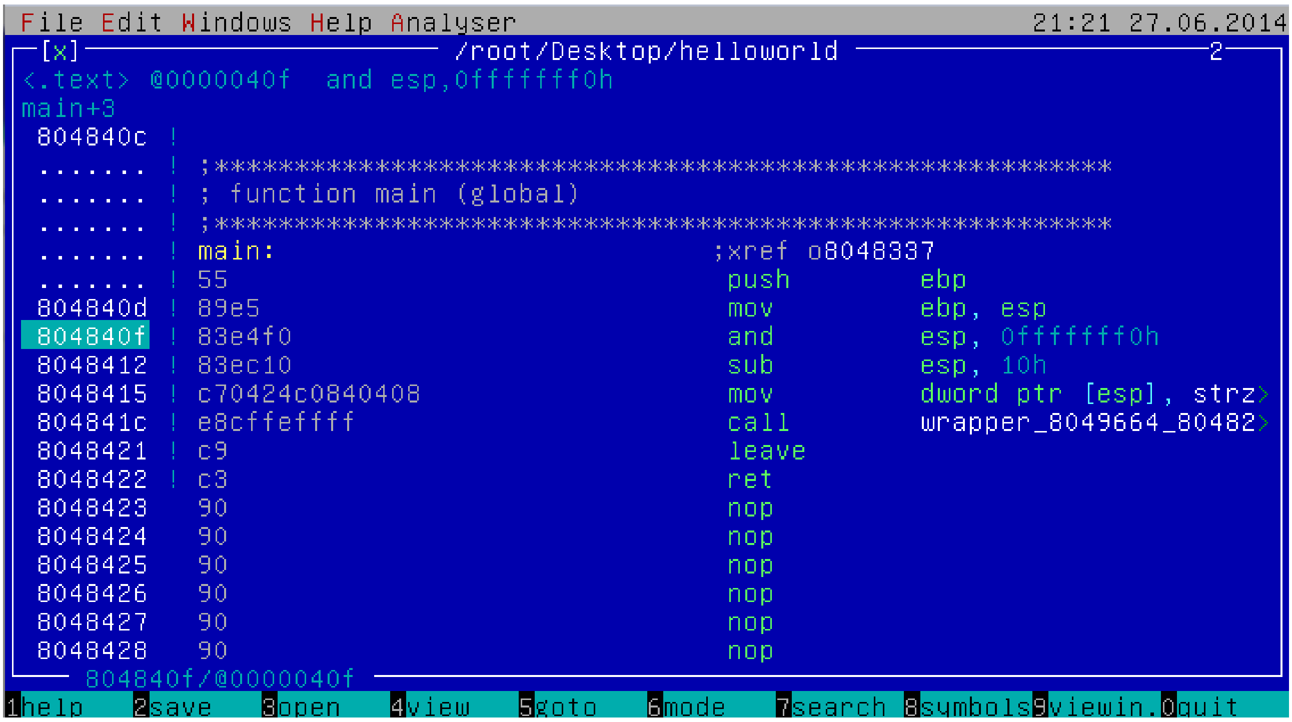


Take a look at the instruction next to 00000000 “jg 0x47” this instruction means jump to the address 0x47 if greater than. But why would even be jumping this early? We haven't even made a comparison yet.

An elf file contains more than just instructions for a program. In fact the actual program instructions are located in a different segment of the program called the .text segment. The diagram to the right shows the different segments of an elf file.

The first 34 bytes in our file are actually part of the ELF file headers and don't contain any instructions written by the programmer. There is a lot of information located here but not instructions. The elf file headers define the entry point of the program, the endianess of the program, and even whether or not the program is a 32 bit or 64 bit executable. Check out the Recommended Resources link 1 for more information.

Lets switch hte's mode into a more advanced mode that will recognize program and section headers. Press the space bar again and select “- elf/image”

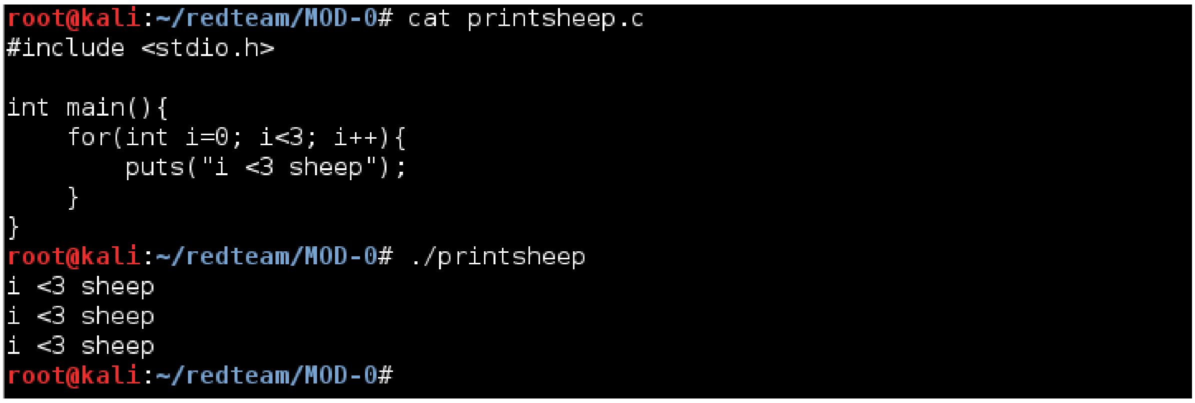


```
 File Edit Windows Help Analyser                                    21:21 27.06.2014
 [x]                      /root/Desktop/helloworld                             2
<.text> @0000040f  and esp,0fffffff0h
main+3
 804840c !
 ....... ! ;*********************************************************
 ....... ! ; function main (global)
 ....... ! ;*********************************************************
 ....... ! main:                             ;xref o8048337
 ....... ! 55                                 push        ebp
 804840d ! 89e5                               mov         ebp, esp
 804840f ! 83e4f0                             and         esp, 0fffffff0h
 8048412 ! 83ec10                             sub         esp, 10h
 8048415 ! c70424c0840408                     mov         dword ptr [esp], strz>
 804841c ! e8cffeffff                         call        wrapper_8049664_80482>
 8048421 ! c9                                 leave
 8048422 ! c3                                 ret
 8048423   90                                 nop
 8048424   90                                 nop
 8048425   90                                 nop
 8048426   90                                 nop
 8048427   90                                 nop
 8048428   90                                 nop
     804840f/@0000040f
1help   2save   3open   4view   5goto   6mode   7search 8symbols9viewin.0quit
```

Wow! This is much better. Not only do we have a labeled <.text> section, but since this ELF file isn't stripped we get a label where the main function begins as well.

You may have also noticed that the hex on the far left is no longer byte numbers in the file. A simple program that prints “hello world this is SIT” is very unlikely to be 134 megabytes. In this mode of hte, the disassembler is showing us where our text segment would be mapped to virtual memory. We will talk more about what virtual memory is in the next module. Keep in mind we haven't ACTUALLY put the program in ram yet since we haven't run the program. Right now the program is just a sequence of bytes in a file.

Before we end this module lets take a look at how we can modify instructions using the ht editor. hte has two different modes for patching binaries. There is the default mode which lets us modify the actual hex values of the binary and there is “assembly mode” that allows us to to actually type in assembly instructions (you can access this with ctrl+a). Lets try this out on a different binary called “printsheep” shown below.

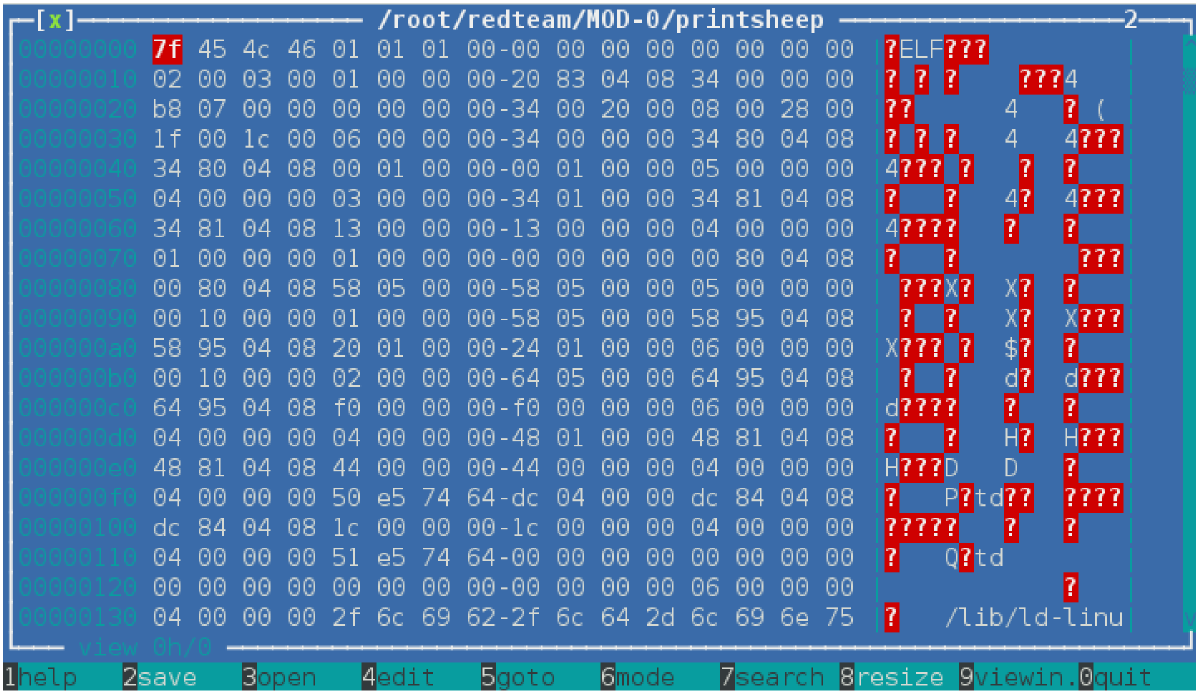


```
root@kali:~/redteam/MOD-0# cat printsheep.c
#include <stdio.h>

int main(){
    for(int i=0; i<3; i++){
        puts("i <3 sheep");
    }
}
root@kali:~/redteam/MOD-0# ./printsheep
i <3 sheep
i <3 sheep
i <3 sheep
root@kali:~/redteam/MOD-0#
```

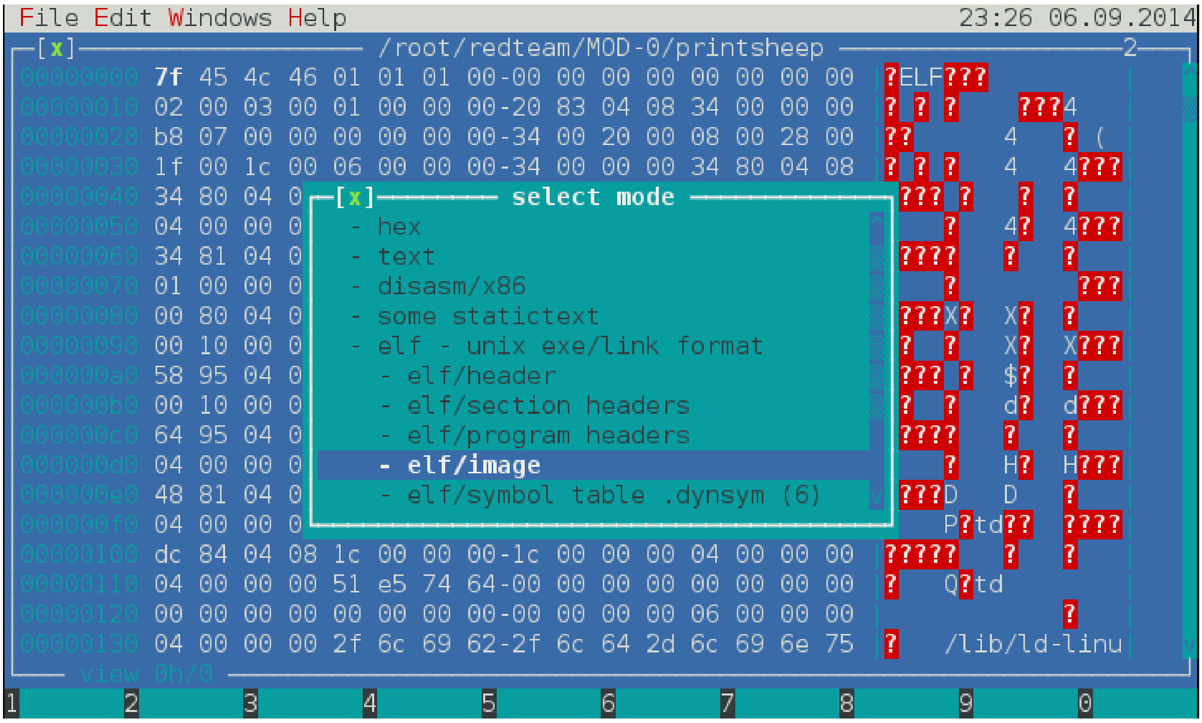
Lets try modifying this binary after its already been compiled. Instead of printing “I <3 sheep” three times lets make it print 10 times.

hte printsheep

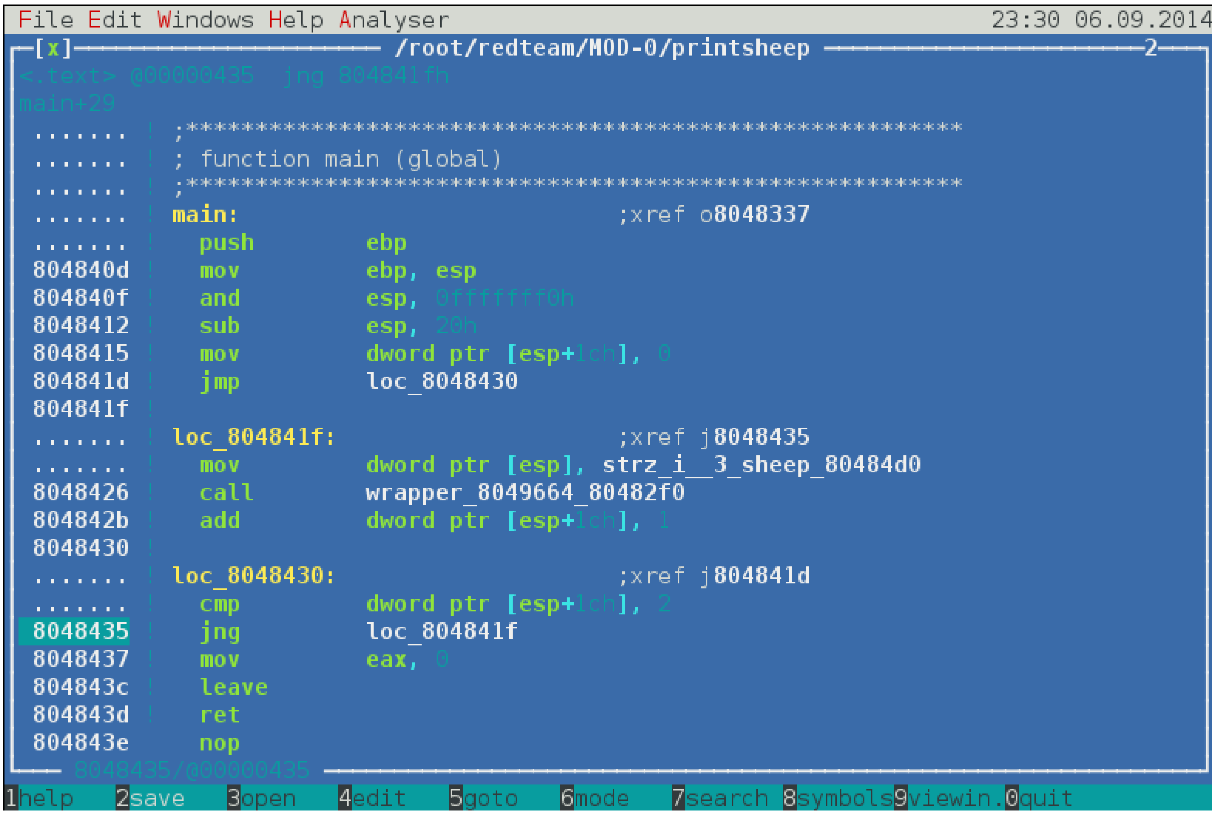


```
[x]──────────────── /root/redteam/MOD-0/printsheep ──────────────2
00000000 7f 45 4c 46 01 01 01 00-00 00 00 00 00 00 00 00 |?ELF???
00000010 02 00 03 00 01 00 00 00-20 83 04 08 34 00 00 00 |? ? ?    ???4
00000020 b8 07 00 00 00 00 00 00-34 00 20 00 08 00 28 00 |??      4   ? (
00000030 1f 00 1c 00 06 00 00 00-34 00 00 00 34 80 04 08 |? ? ?   4   4???
00000040 34 80 04 08 00 01 00 00-00 01 00 00 05 00 00 00 |4??? ?   ?   ?
00000050 04 00 00 00 03 00 00 00-34 01 00 00 34 81 04 08 |?   ?   4?  4???
00000060 34 81 04 08 13 00 00 00-13 00 00 00 04 00 00 00 |4????   ?   ?
00000070 01 00 00 00 01 00 00 00-00 00 00 00 00 80 04 08 |?   ?        ???
00000080 00 80 04 08 58 05 00 00-58 05 00 00 05 00 00 00 | ???X?  X?  ?
00000090 00 10 00 00 01 00 00 00-58 05 00 00 58 95 04 08 | ?  ?   X?  X???
000000a0 58 95 04 08 20 01 00 00-24 01 00 00 06 00 00 00 |X??? ?  $?  ?
000000b0 00 10 00 00 02 00 00 00-64 05 00 00 64 95 04 08 | ?  ?   d?  d???
000000c0 64 95 04 08 f0 00 00 00-f0 00 00 00 06 00 00 00 |d????   ?   ?
000000d0 04 00 00 00 04 00 00 00-48 01 00 00 48 81 04 08 |?   ?   H?  H???
000000e0 48 81 04 08 44 00 00 00-44 00 00 00 04 00 00 00 |H???D   D   ?
000000f0 04 00 00 00 50 e5 74 64-dc 04 00 00 dc 84 04 08 |?   P?td??  ????
00000100 dc 84 04 08 1c 00 00 00-1c 00 00 00 04 00 00 00 |?????   ?   ?
00000110 04 00 00 00 51 e5 74 64-00 00 00 00 00 00 00 00 |?   Q?td
00000120 00 00 00 00 00 00 00 00-00 00 00 00 06 00 00 00 |            ?
00000130 04 00 00 00 2f 6c 69 62-2f 6c 64 2d 6c 69 6e 75 |?   /lib/ld-linu
──── view 0h/0 ─────────────────────────────────────────────────────
1help 2save 3open 4edit 5goto 6mode 7search 8resize 9viewin.0quit
```

For changing instructions, its much easier to edit things from image mode. Lets go ahead and change into that using spacebar.



```
File Edit Windows Help                                   23:26 06.09.2014
[x]──────────────── /root/redteam/MOD-0/printsheep ──────────────2
00000000 7f 45 4c 46 01 01 01 00-00 00 00 00 00 00 00 00 |?ELF???
00000010 02 00 03 00 01 00 00 00-20 83 04 08 34 00 00 00 |? ? ?    ???4
00000020 b8 07 00 00 00 00 00 00-34 00 20 00 08 00 28 00 |??      4   ? (
00000030 1f 00 1c 00 06 00 00 00-34 00 00 00 34 80 04 08 |? ? ?   4   4???
00000040 34 80 04 0 ┌[x]──────── select mode ──────────┐??? ?   ?   ?
00000050 04 00 00 0 │ - hex                           │    ?   4?  4???
00000060 34 81 04 0 │ - text                          │????   ?   ?
00000070 01 00 00 0 │ - disasm/x86                    │    ?        ???
00000080 00 80 04 0 │ - some statictext               │???X?  X?  ?
00000090 00 10 00 0 │ - elf - unix exe/link format    │?   ?   X?  X???
000000a0 58 95 04 0 │   - elf/header                  │??? ?  $?  ?
000000b0 00 10 00 0 │   - elf/section headers         │?   ?   d?  d???
000000c0 64 95 04 0 │   - elf/program headers         │????   ?   ?
000000d0 04 00 00 0 │   - elf/image                   │    ?   H?  H???
000000e0 48 81 04 0 │   - elf/symbol table .dynsym (6)│???D   D   ?
000000f0 04 00 00 0 └─────────────────────────────────┘    P?td??  ????
00000100 dc 84 04 08 1c 00 00 00-1c 00 00 00 04 00 00 00 |?????   ?   ?
00000110 04 00 00 00 51 e5 74 64-00 00 00 00 00 00 00 00 |?   Q?td
00000120 00 00 00 00 00 00 00 00-00 00 00 00 06 00 00 00 |            ?
00000130 04 00 00 00 2f 6c 69 62-2f 6c 64 2d 6c 69 6e 75 |?   /lib/ld-linu
──── view 0h/0 ─────────────────────────────────────────────────────
1       2       3       4       5       6       7       8       9       0
```

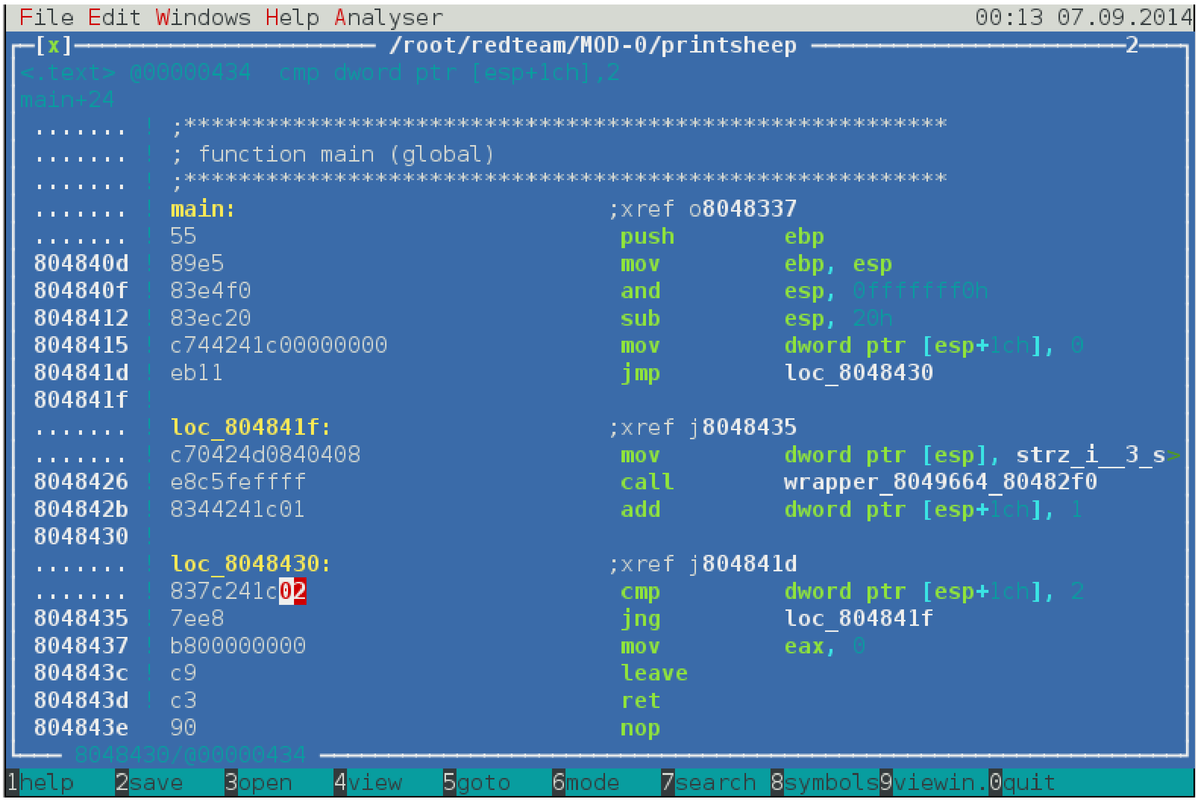


```
 File Edit Windows Help Analyser                                    23:30 06.09.2014
┌─[x]──────────────────── /root/redteam/MOD-0/printsheep ─────────────────────2──┐
│<.text> @00000435  jng 804841fh
│main+29
│ .......  ! ;****************************************************
│ .......  ! ; function main (global)
│ .......  ! ;****************************************************
│ .......  ! main:                          ;xref o8048337
│ .......  !   push        ebp
│ 804840d  !   mov         ebp, esp
│ 804840f  !   and         esp, 0fffffff0h
│ 8048412  !   sub         esp, 20h
│ 8048415  !   mov         dword ptr [esp+1ch], 0
│ 804841d  !   jmp         loc_8048430
│ 804841f  !
│ .......  ! loc_804841f:                   ;xref j8048435
│ .......  !   mov         dword ptr [esp], strz_i__3_sheep_80484d0
│ 8048426  !   call        wrapper_8049664_80482f0
│ 804842b  !   add         dword ptr [esp+1ch], 1
│ 8048430  !
│ .......  ! loc_8048430:                   ;xref j804841d
│ .......  !   cmp         dword ptr [esp+1ch], 2
│ 8048435  !   jng         loc_804841f
│ 8048437  !   mov         eax, 0
│ 804843c  !   leave
│ 804843d  !   ret
│ 804843e      nop
└──── 8048435/@00000435 ─────────────────────────────────────────────────────────┘
1help  2save   3open   4edit   5goto   6mode   7search 8symbols9viewin.0quit
```

You may have to scroll down (using page down key) until you find the main function like in the picture above.

The trick now is to actually figure out what the assembly does. If this is your first time actually taking a look at assembly the entire process can be incredibly overwhelming. Infact quite a few of the instructions may not make sense until after RED-MOD-1 where we take a deeper look at memory. To help you out we've included a file that annotates every line of the assembly. In this case to get “I <3 sheep” to print 10 times we have to change the instruction cmp dword ptr [esp+1ch], 2 to cmp dword ptr [esp+1ch], 0ah

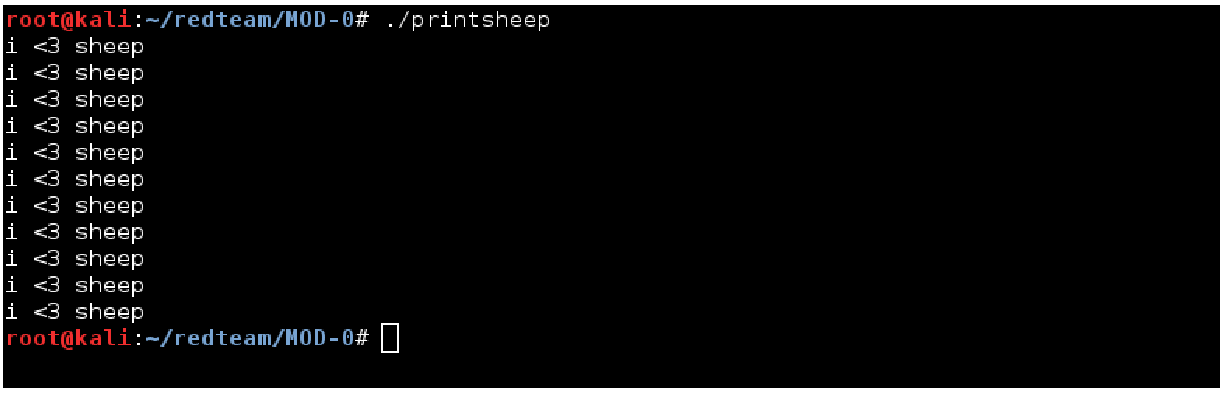
To do this press the F4 key to edit the hex.



File Edit Windows Help Analyser
00:13 07.09.2014
[x]
/root/redteam/MOD-0/printsheep
2
1help 2save 3open 4view 5goto 6mode 7search 8symbols9viewin.0quit

```
<.text> @00000434  cmp dword ptr [esp+1ch],2
main+24
 .......  ! ;*****************************************************
 .......  ! ; function main (global)
 .......  ! ;*****************************************************
 .......  ! main:                            ;xref o8048337
 .......  ! 55                                push        ebp
 804840d  ! 89e5                              mov         ebp, esp
 804840f  ! 83e4f0                            and         esp, 0fffffff0h
 8048412  ! 83ec20                            sub         esp, 20h
 8048415  ! c744241c00000000                  mov         dword ptr [esp+1ch], 0
 804841d  ! eb11                              jmp         loc_8048430
 804841f  !
 .......  ! loc_804841f:                     ;xref j8048435
 .......  ! c70424d0840408                    mov         dword ptr [esp], strz_i__3_s>
 8048426  ! e8c5feffff                        call        wrapper_8049664_80482f0
 804842b  ! 8344241c01                        add         dword ptr [esp+1ch], 1
 8048430  !
 .......  ! loc_8048430:                     ;xref j804841d
 .......  ! 837c241c02                        cmp         dword ptr [esp+1ch], 2
 8048435  ! 7ee8                              jng         loc_804841f
 8048437  ! b800000000                        mov         eax, 0
 804843c  ! c9                                leave
 804843d  ! c3                                ret
 804843e    90                                nop
 8048430/@00000434
```

We need to change the very last opcode from 02 to 0a the hex equivalent of 10. Then, push F2 to save the file. Now when we run printsheep we should get the output below.



```
root@kali:~/redteam/MOD-0# ./printsheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
i <3 sheep
root@kali:~/redteam/MOD-0# 
```

Exercises:

CHHHHHHALLENGE MODE – Work with your friends at your table to try and solve these entry level binary challenges. As you move down the list the challenges will get harder!

Future Application:

In the next Module we will begin writing our own exploits for systems that use the x86 architecture. In order to understand whats going on while exploiting a system, its critical to develop an understanding of what a program is.

Recommended Resources:

http://en.wikipedia.org/wiki/Executable_and_Linkable_Format
http://sparksandflames.com/files/x86InstructionChart.html

This Module was written by Vincent Moscatello for the Organization: Student Infosec Team.