Debugging Linux Applications

jims@bluenotch.com

Today's Agenda

- Debugging Linux Applications
  - Application background
    - Introduction to ELF
    - Loaders and Linkers
  - Tools
    - gcc and gdb
    - objdump
    - Others (file, strings, systrace/apptrace/dtrace)

© 2008 James Shewmaker

Generally How to Stay Sane

- Sometimes it is best to change your perspective while debugging
  - Sometimes that means going literally outside the box
  - Sometimes that means getting in between the little grooves of cardboard
- Log files help, debug printf's everywhere can help but can also lead you to bad assumptions.

© 2008 James Shewmaker

i386 briefly

- Registers
  - EAX, EBX, EDX, ESX, EBX
  - Memory and the stack

© 2008 James Shewmaker

Introduction to ELF

- Classic a.out vs. ELF
  - a.out=jumptables; ELF allows dynamic shared libraries
  - No longer in FreeBSD 7.X and Linux kernel 2.6.25
- Loaders and Linkers
  - Loaders put a program into memory
  - Linkers deal with attaching symbols to memory addresses
  - Either can relocate (adjust for non-overlapping

© 2008 James Shewmaker

Example: fact.c

- Computes factorials from user input

```
jim@demo:~$ ./fact
Enter the value:12
You entered:
12
The factorial of 12 is 0
jim@demo:~$ file fact
fact: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.8, dynamically linked (uses shared libs), not stripped
```


© 2008 James Shewmaker

ojbdump –t ./fact (1)

08048708 l d .eh_frame 00000000 .eh_frame

0804970c l d .ctors 00000000 .ctors

08049714 l d .dtors 00000000 .dtors

0804971c l d .jcr 00000000 .jcr

08049720 l d .dynamic 00000000 .dynamic

080497f0 l d .got 00000000 .got

080497f4 l d .got.plt 00000000 .got.plt

08049824 l d .data 00000000 .data

08049840 l d .bss 00000000 .bss

© 2008 James Shewmaker

objdump –t ./fact (2)

080498c4 g O .bss 00000004 factn

0804856f g F .text 0000003b buildResultString

080484d4 g F .text 00000073 getInt

080498c8 g O .bss 00000004 n

080498cc g *ABS* 00000000 _end

00000000 F *UND* 0000018f puts@@GLIBC_2.0

08049830 g *ABS* 00000000 _edata

08048547 g F .text 00000028 computeFact

© 2008 James Shewmaker

Getting started with gdb

```
jim@demo:~$ gdb -q ./fact
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) break main
Breakpoint 1 at 0x80485bb: file fact.c, line 67.
(gdb) run
Starting program: /home/jim/fact
Breakpoint 1, main () at fact.c:67
8	n = getInt();
(gdb) step
getInt () at fact.c:26
26		inputString = (char *) malloc (MAXINPUTSTRINGSIZE *
	sizeof(char));
```


© 2008 James Shewmaker

Continue with gdb

```
(gdb) break computeFact
Breakpoint 2 at 0x804854d: file fact.c, line 43.
(gdb) cont
Continuing.
Enter the value:5
You entered:
5
Breakpoint 2, computeFact (n=5) at fact.c:43
9   int accum=0;
```


© 2008 James Shewmaker

Finding the bug

```
(gdb) step
45              while(n>1) {
(gdb) step
46                      accum *= n;
(gdb) print accum
$1 = 0
(gdb) step
47                      n--;
(gdb) step
45              while(n>1) {
```


© 2008 James Shewmaker

Disassembly of computeFact

```
0x08048547 <computeFact+0>:     push   %ebp
0x08048548 <computeFact+1>:     mov    %esp,%ebp
0x0804854a <computeFact+3>:     sub    $0x10,%esp
0x0804854d <computeFact+6>:     movl   $0x0,0xfffffffc(%ebp)
0x08048554 <computeFact+13>:    jmp    0x8048564 <computeFact+29>
0x08048556 <computeFact+15>:    mov    0xfffffffc(%ebp),%eax
0x08048559 <computeFact+18>:    imul   0x8(%ebp),%eax
0x0804855d <computeFact+22>:    mov    %eax,0xfffffffc(%ebp)
0x08048560 <computeFact+25>:    subl   $0x1,0x8(%ebp)
0x08048564 <computeFact+29>:    cmpl   $0x1,0x8(%ebp)
0x08048568 <computeFact+33>:    jg     0x8048556 <computeFact+15>
0x0804856a <computeFact+35>:    mov    0xfffffffc(%ebp),%eax
0x0804856d <computeFact+38>:    leave
0x0804856e <computeFact+39>:    ret
```


© 2008 James Shewmaker

Keeping track of where you are at

```
(gdb) display n
1: n = 3
(gdb) info stack
#0  computeFact (n=3) at fact.c:45
#1  0x080485d2 in main () at fact.c:68
(gdb) s
46                              accum *= n;
1: n = 3
```


© 2008 James Shewmaker

Summary

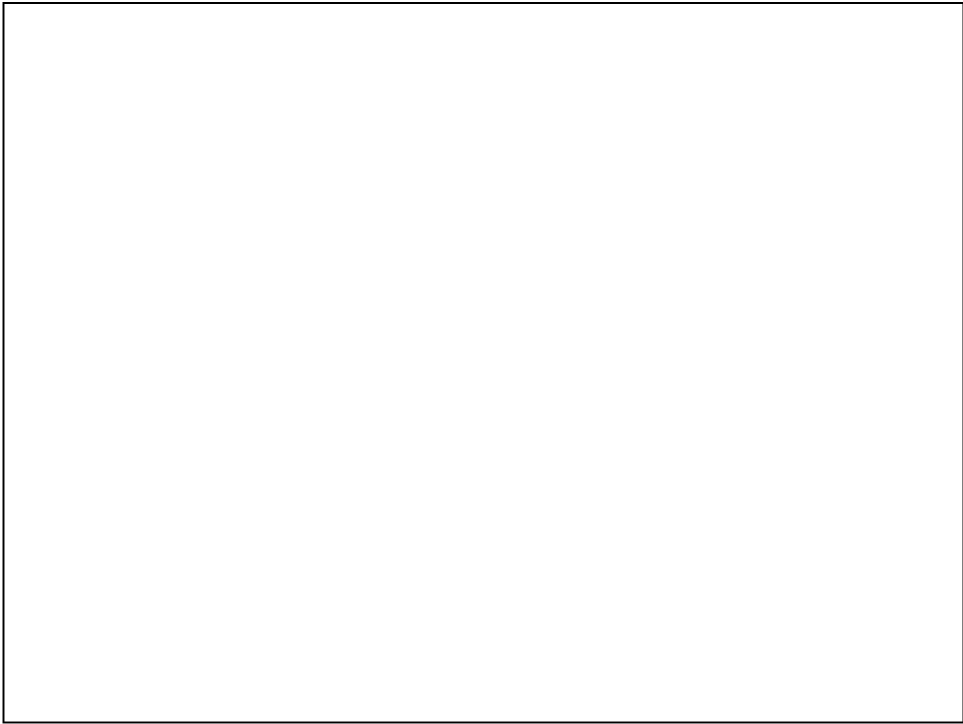
- You must get your hands dirty if you want to learn
- man pages aren't very useful—too many exceptions to every rule
- Google a bit, but experiment on a sample.
  - Then sample on a different compiler/distribution/CPU
- Even the easy stuff can be confusing, start with what you know and build into more complication

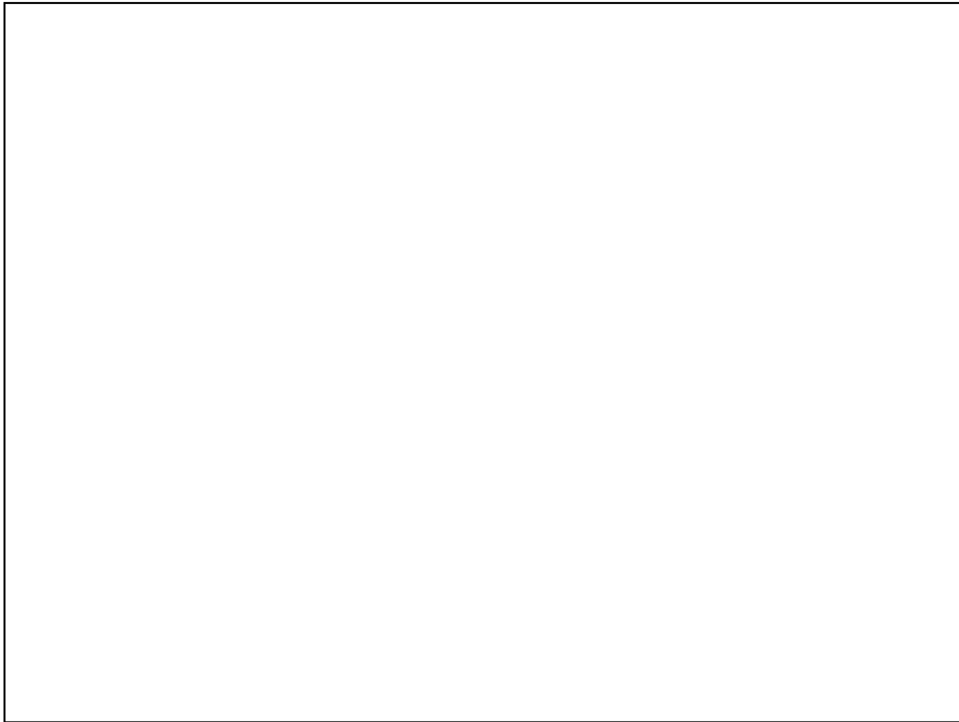
© 2008 James Shewmaker

References / For More Information

- **http://en.wikipedia.org/wiki/Executable_and_Linkable_Format**
- **“Linkers and Loaders” by John Levine**
- **fact.c: http://www.cs.dartmouth.edu/~dwagn/cs58/tutorial/fact1.c**
- **Beginner notes on reverse engineering in FreeBSD: http://wantingseed.com/raw/TheMakingOfAtlas-dc-06.pdf**
- **Using GDB: http://sourceware.org/gdb/download/onlinedocs/gdb_toc.html#SEC_Contents**
- **Recursive Reference: http://bluenotch.com/resources/DebuggingLinuxApps.pdf**

© 2008 James Shewmaker





The purpose of today's presentation is to cover some general tips of debugging Linux (well, ELF binary programs at large, not just Linux) applications. Because I am involved heavily in the security community, these troubleshooting tips will come from security related examples. The tips themselves are useful for everyday troubleshooting—I just prefer to use examples that also demonstrate the real reason vulnerable code is a very bad thing.

This material is an introduction into debugging, taken from random examples I have used to demonstrate a compiler induced or other glitch. There are many tutorials out there that can provide more examples, but I try to demystify some of what you'll run into when googling for these types of glitches.

We will focus on common tools you probably already have laying around on your Linux system, these examples are demonstrated on Ubuntu 7.10 (Linux kernel 2.6.22-14 which has gcc 4.1.3) although they should work on any semi-recent operating system. On any given system the memory locations might not be exactly what I have here, but the examples are not complex enough to deviate much. Most of our time will be spent in gdb, but we will use a few other tools to help us find out what may be wrong with our application. The objdump and readelf tools will really help in dissecting a sample and figure out what is really going on. Sometimes, other utilities will help (especially with a malformed sample program) by showing a glitch in the structure or another clue that could lead us to the proper understanding of what is going wrong. The commands I use all the time for sanity checks like these are file and strings.

Generally How to Stay Sane

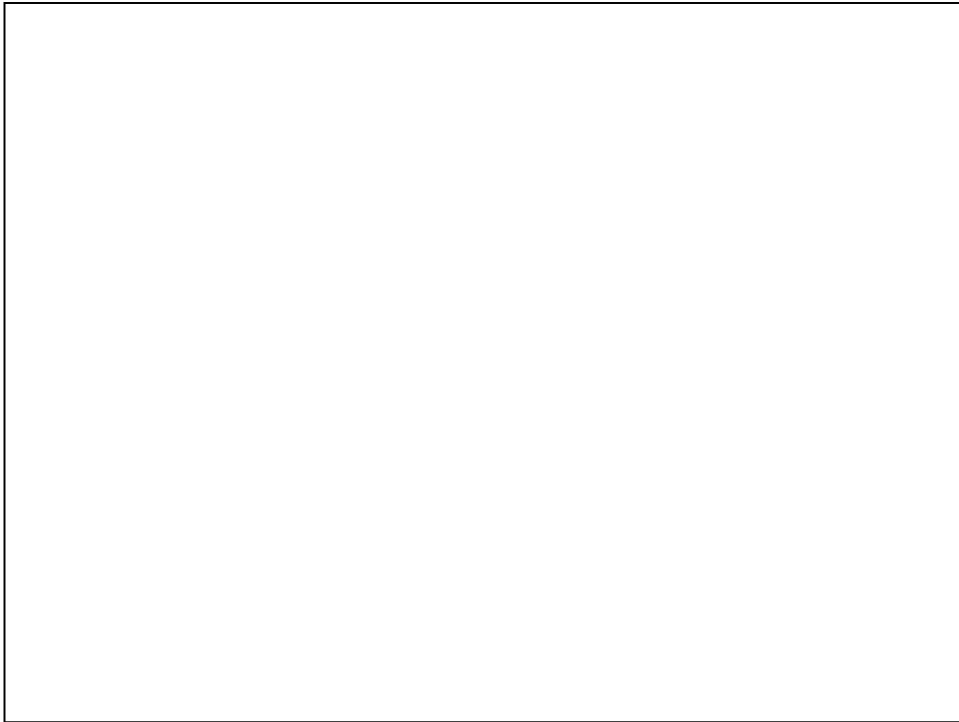
- Sometimes it is best to change your perspective while debugging
  - Sometimes that means going literally outside the box
  - Sometimes that means getting in between the little grooves of cardboard
- Log files help, debug printf's everywhere can help but can also lead you to bad assumptions.

© 2008 James Shewmaker

i386 briefly

- Registers
  - EAX, EBX, EDX, ESX, EBX
  - Memory and the stack

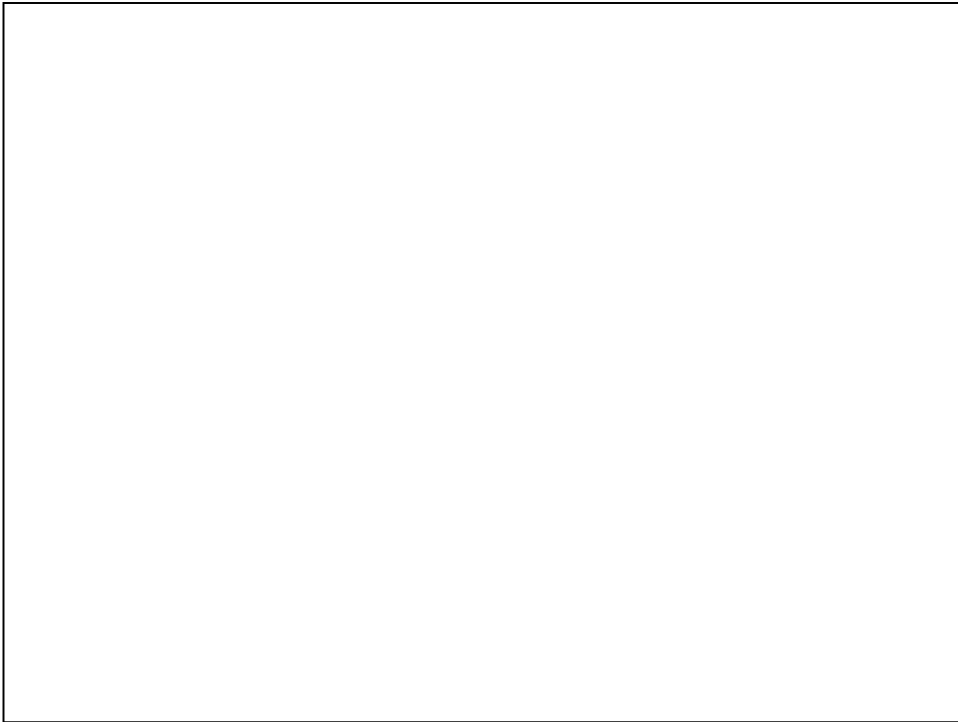
© 2008 James Shewmaker



For today, we will be focusing on ELF binary executables. Once upon a time, everyone with a POSIX UNIX-like system used an “a.out” format that was statically compiled with everything necessary to run contained inside the same binary file. The a.out format used jumptables to keep track of which functions were where, and was very simple but did not offer any extra features for cross-platform compiling and dynamic libraries to be shared with other executable programs.

ELF (Executable and Linkable Format) was finalized in 1995. Some of the UNIX-like operating systems didn’t immediately adopt the standard, FreeBSD didn’t start supporting ELF until the 3.X branch. FreeBSD 7.0-RELEASE removed the last remaining support for a.out. Linux kernel 2.6.25 should also remove the last traces of a.out, mostly in an abstraction layer that shimmed in some support of a.out in an ELF world. Also, don’t assume that you compiled a binary executable as an a.out type just because the compiler decided to name the result a.out. Look at your result

If you are interested in the Linking and Loading of different types of executables for different operating systems, the book “Linkers and Loaders” by John R. Levine is the best book to start with. Each executable program can also show subtle differences depending on the compiler, compiler version, and command-line flags given to the compiler. All of these things may complicate your troubleshooting efforts, so try and simplify the test as much as you can before you start getting creative. We won’t cover the linking and loading too much here—I just wanted to mention them for background information.



Here we use a factorial sample program with a bug in it (See references for the source and author). This program has a few functions in it, accepts user input and provides output (incorrect results) back to standard out. The file command shows us some information about the file itself. The last thing it tells us, “not stripped” is valuable for us. This means that the symbols used in the program have not been pulled out for efficiency or obfuscation.

ojbdump –t ./fact (1)

08048708 l d .eh_frame 00000000 .eh_frame

0804970c l d .ctors 00000000 .ctors

08049714 l d .dtors 00000000 .dtors

0804971c l d .jcr 00000000 .jcr

08049720 l d .dynamic 00000000 .dynamic

080497f0 l d .got 00000000 .got

080497f4 l d .got.plt 00000000 .got.plt

08049824 l d .data 00000000 .data

08049840 l d .bss 00000000 .bss

© 2008 James Shewmaker

objdump –t ./fact (2)

```
080498c4 g     O .bss   00000004              factn
0804856f g     F .text  0000003b
    buildResultString
080484d4 g     F .text  00000073              getInt
080498c8 g     O .bss   00000004              n
080498cc g       *ABS*  00000000              _end
00000000       F *UND*  0000018f
    puts@@GLIBC_2.0
08049830 g       *ABS*  00000000              _edata
08048547 g     F .text  00000028              computeFact
```


© 2008 James Shewmaker

Getting started with gdb

```
jim@demo:~$ gdb -q ./fact
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) break main
Breakpoint 1 at 0x80485bb: file fact.c, line 67.
(gdb) run
Starting program: /home/jim/fact
Breakpoint 1, main () at fact.c:67
8    n = getInt();
(gdb) step
getInt () at fact.c:26
26          inputString = (char *) malloc (MAXINPUTSTRINGSIZE *
     sizeof(char));
```


© 2008 James Shewmaker

Continue with gdb

```
(gdb) break computeFact
Breakpoint 2 at 0x804854d: file fact.c, line 43.
(gdb) cont
Continuing.
Enter the value:5
You entered:
5
Breakpoint 2, computeFact (n=5) at fact.c:43
9   int accum=0;
```


© 2008 James Shewmaker

Finding the bug

```
(gdb) step
45              while(n>1) {
(gdb) step
46                      accum *= n;
(gdb) print accum
$1 = 0
(gdb) step
47                      n--;
(gdb) step
45              while(n>1) {
```


© 2008 James Shewmaker

Disassembly of computeFact

```
0x08048547 <computeFact+0>:     push   %ebp
0x08048548 <computeFact+1>:     mov    %esp,%ebp
0x0804854a <computeFact+3>:     sub    $0x10,%esp
0x0804854d <computeFact+6>:     movl   $0x0,0xfffffffc(%ebp)
0x08048554 <computeFact+13>:    jmp    0x8048564 <computeFact+29>
0x08048556 <computeFact+15>:    mov    0xfffffffc(%ebp),%eax
0x08048559 <computeFact+18>:    imul   0x8(%ebp),%eax
0x0804855d <computeFact+22>:    mov    %eax,0xfffffffc(%ebp)
0x08048560 <computeFact+25>:    subl   $0x1,0x8(%ebp)
0x08048564 <computeFact+29>:    cmpl   $0x1,0x8(%ebp)
0x08048568 <computeFact+33>:    jg     0x8048556 <computeFact+15>
0x0804856a <computeFact+35>:    mov    0xfffffffc(%ebp),%eax
0x0804856d <computeFact+38>:    leave
0x0804856e <computeFact+39>:    ret
```


© 2008 James Shewmaker

Keeping track of where you are at

```
(gdb) display n
1: n = 3
(gdb) info stack
#0  computeFact (n=3) at fact.c:45
#1  0x080485d2 in main () at fact.c:68
(gdb) s
46                    accum *= n;
1: n = 3
```


© 2008 James Shewmaker

Summary

- You must get your hands dirty if you want to learn
- man pages aren’t very useful—too many exceptions to every rule
- Google a bit, but experiment on a sample.
  - Then sample on a different compiler/distribution/CPU
- Even the easy stuff can be confusing, start with what you know and build into more complication

© 2008 James Shewmaker

References / For More Information

- **http://en.wikipedia.org/wiki/Executable_and_Linkable_Format**
- **“Linkers and Loaders” by John Levine**
- **fact.c: http://www.cs.dartmouth.edu/~dwagn/cs58/tutorial/fact1.c**
- **Beginner notes on reverse engineering in FreeBSD: http://wantingseed.com/raw/TheMakingOfAtlas-dc-06.pdf**
- **Using GDB: http://sourceware.org/gdb/download/onlinedocs/gdb_toc.html#SEC_Contents**
- **Recursive Reference: http://bluenotch.com/resources/DebuggingLinuxApps.pdf**

© 2008 James Shewmaker