Day 7
Debugging

When bad things happen to good programs

Joel Isaacson
Ascender Technologies Ltd.
http://ascender.com

Today's Lectures

- Today we will discuss what tools are useful in debugging programs.
- Programs don't debug programs. People debug programs.
- Knowing what question to ask is crucial.
- We will discuss tools that can be used understand program execution.

Malloc/Free Problems

- Malloc/Free is undoubtedly the most problematical function in embedded systems.
- Each Malloc has to have a Free call.
- To many Free's and the application crashes.
- To few Free's and you have a storage leak.

Allocating New Memory

- Linux has two ways to allocate new memory.
  - brk/sbrk: This is the traditional Unix method of allocating memory. It extends the data segment.
  - mmap: This method of allocating memory first was implemented in BSD Unix.

brk - sbrk

Change Data Segment Size

- `brk(eds)` sets the end of the data segment to the value specified by eds, when that value is reasonable, the system does have enough memory and the process does not exceed its max data size.
- `sbrk(len)` extends the end of the data segment by len.

brk - sbrk
Change Data Segment Size

```
#include <unistd.h>

int main(int argc, char *argv[])
{
    while(1) {
        sbrk(4*1024);
        sleep(1);
    }
}
```

mmap
Allocate Memory Segment

- `mmap()` - mmap can be used to allocate, in the virtual address space of the calling process, an anonymous memory segment that is not mapped to any file.
- This segment is initialized to zeros.

mmap()
Allocate Data Segment

```
#include <sys/mman.h>
int main(int argc, char *argv[])
{
  while(1) {
    mmap(0, 4*1024,
         PROT_WRITE | PROT_READ,
         MAP_PRIVATE |MAP_ANONYMOUS,
         0, 0);
    sleep(1);
  }
}
```

Malloc/Mtrace

- This is the standard memory debugger included in the GNU C Library.
- The function mtrace installs handlers for malloc, realloc and free; the function muntrace disables these handlers.
- A perl script also called mtrace parses through the output file and reports all allocations that were not freed.

Mtrace

- The handlers log all memory allocations and frees to a file defined by the environment variable MALLOC_TRACE.

```
MALLOC_TRACE=mytracefile
export MALLOC_TRACE
cc -g -o mt mt.c
mtrace mt mytracefile
```

mt.c

```
#include <stdlib.h>
#include <mcheck.h>
// export MALLOC_TRACE=mt.trace
int main() {
   mtrace(); /* Starts the recording */
   int* a = NULL;
   a = malloc(sizeof(int)); /* allocate memory */
   if (a == NULL) {
       return 1; /* error */
   }
   free(a); /* we free the memory */
   muntrace();
   return 0; /* exit */
}
```

Memory Leaks

- If malloc'ed memory isn't freed and is reallocated frequently then we have a “memory leak”.
- Memory leaks are particularly problematical in embedded systems with limited resources and no swap space.

top

- The top program provides a dynamic real-time view of a running system.
- It can display system information as well as a list of tasks currently being managed by the Linux kernel.
- The 'M' option will sort the list of processes by memory utilization.

Electric Fence

- Electric Fence helps you detect two common programming bugs: software that overruns the boundaries of a `malloc()` memory allocation, and software that touches a memory allocation that has been released by `free()`.

Electric Fence

- Unlike other malloc() debuggers, Electric Fence will detect read accesses as well as writes, and it will pinpoint the exact instruction that causes an error.
- Electric Fence usually is run in conjunction with a debugger (gdb).

Electric Fence

- Electric Fence uses the virtual memory hardware to place an inaccessible memory page immediately after (or before, at the user's option) each memory allocation.
- When software reads or writes this inaccessible page, the hardware issues a segmentation fault, stopping the program at the offending instruction.

Electric Fence

- It is then trivial to find the erroneous statement using a debugger.
- In a similar manner, memory that has been released by free() is made inaccessible, and any code that touches it will get a segmentation fault.

Electric Fence

- Simply linking your application with libefence.a will allow you to detect most, but not all, malloc buffer overruns and accesses of free memory.
- Electric Fence usually greatly overuses virtual memory resources.
- It should not be used in production software.

Electric Fence - ef.c

```
#include <stdlib.h>
#include <string.h>
#define SIZE 16

int main() {
        int i;
        char *cp=malloc(SIZE);
        for(i=0;i<=SIZE;i++)
                cp[i]= 0;
}
```

Electric Fence - ef1.c

```
#include <stdlib.h>
#include <string.h>
#define SIZE 16

int main() {
        int i;
        char *cp=malloc(SIZE);
        for(i=0;i<=SIZE;i++)
                cp[i-1]= 0;
}
```

Electric Fence - ef2.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
extern int EF_ALIGNMENT, EF_PROTECT_BELOW;
int main() {
        int i;
        // EF_ALIGNMENT=0;
        // EF_PROTECT_BELOW=1;
        for(i=0;i<12;i++)
                printf("%p\n", malloc(i+1));
}
```

Electric Fence - ef3.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
extern int EF_PROTECT_FREE;
int main() {
        char *cp, *cp2;
        // EF_PROTECT_FREE= 1;
        cp= malloc(10);
        free(cp);
        cp2= malloc(10);
        *cp=0;
}
```

Electric Fence - ef4.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
extern int EF_PROTECT_FREE;
int main() {
        int i;
        // EF_PROTECT_FREE= 1;
        for(i=0;i<12;i++) {
                char *cp;
                printf("%p\n", cp= malloc(i+1));
                free(cp);
        }
}
```

catchsegv

- Used to debug segmentation faults in programs.
- The output is the content of registers, plus a backtrace.
- Basically you call your program and its arguments as the arguments to catchsegv.

catchsegv - LD_PRELOAD

- Catchsegv in itself is interesting but the technique that it uses to dynamically add arbitrary executable code to a linked program is by itself useful.
- The following command will give the effect of catchsegv:

```
LD_PRELOAD=/lib/libSegFault.so ./ef
```

Use of `strcmp`

```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
  char passwd[] = "foobar";
  if (argc < 2) {
    printf("usage: %s <given-password>\n", argv[0]);
    return 0;
  }
  if (!strcmp(passwd, argv[1])) {
    printf("Green light!\n");
    return 1;
  }
  printf("Red light!\n");
  return 0;
}
```

Dynamic preemption of `strcmp`

`strcmp-hijack.c`

```
#define _GNU_SOURCE
#include <stdio.h>
#include <string.h>
#include <dlfcn.h>

static int (*_strcmp)(const char *s1, const char *s2);

int strcmp(const char *s1, const char *s2) {
  if (_strcmp == NULL) {
    _strcmp = (int (*)(const char *s1, const char *s2))
        dlsym(RTLD_NEXT, "strcmp");
  }
  printf("S1,S2 %s,%s\n", s1, s2);
  return (*_strcmp)(s1, s2);
}
```

Dynamic preemption of `strcmp`

`strcmp-hijack.so`

```
cc -fPIC -c strcmp-hijack.c -o strcmp-hijack.o
cc -shared -o strcmp-hijack.so strcmp-hijack.o -ldl
```

strace

- strace is a useful diagnostic, instructional, and debugging tool.
- It is invaluable for solving problems with programs for which the source is not readily available since they do not need to be recompiled in order to trace them.
- A great deal can be learned about a system and its system calls by tracing even ordinary programs.

strace

- Since system calls and signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions.
- Strace shows you what actually is happening, rather than what you think is happening

ltrace

- ltrace is a program that simply runs the specified command until it exits.
- It intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process.
- It can also intercept and print the system calls executed by the program.

ltrace

- Ltrace works by inserting breakpoints into the child program via ptrace for each traces routine.
- The file /etc/ltrace.conf gives a list of routines that are to be traced.
- Ltrace can be used to analyze performance:

```
ltrace -c -ttt ls -l
```

Core Dumps

- You can instruct Linux to create a file on abnormal termination.
- This file is normally called 'core'.
- This file can be passed to gdb to determine where and why the program failed.
- Under the bash shell “ulimit” is used to allow the core file to be written.

Ptrace

- The ptrace() system call provides a means by which a parent process may observe and control the execution of another process, and examine and change its core image and registers.
- It is primarily used to implement breakpoint debugging and system call tracing.

Ptrace

- The parent can initiate a trace by calling fork(2) and having the resulting child do a PTRACE_TRACEME, followed (typically) by an exec(3).
- Alternatively, the parent may commence trace of an existing process using PTRACE_ATTACH.

Ptrace

- While being traced, the child will stop each time a signal is delivered, even if the signal is being ignored.
- The parent will be notified at its next wait and may inspect and modify the child process while it is stopped.
- The parent then causes the child to continue, optionally ignoring the delivered signal.

Ptrace

- You can also arraign for ptrace to stop the child on system calls.
- This is how strace is implemented.
- Gdb puts an illegal instruction at every breakpoint. To continue execution the illegal instruction is replaced with the original instruction.

Gdb

- Gdb is the premiere debugger for Linux/ GNU.
- It really is very solid.
- It has various graphical front ends, like ddd and insight.
- It pays to learn how to use gdb directly.

Gdbserver

- The gdbserver allows remote debugging via serial line or Tcp/Ip connections.
- This can easily be done between machines of different architectures.
- Remote:
  - gdbserver :port program
- Local:
  - target remote host:port

Kernel Debugging

- Debugging the kernel is a whole other bag.
- It requires much more specialized knowledge.
- In general, kernel hackers eschew sophisticated tools and make do with simple printk's.

Kernel Debugging - Printk

- Printk's in the kernel can be read in various ways:
  - They go the the systems console
  - They can be read with dmesg
  - They are routed to /var/log via syslog

Kernel Debugging - /proc

- In many situations creating a /proc filesystem entry is more useful than printk.
- It can be less intrusive than printk and more responsive to the person that wants to debug the kernel.