Systems Programming

Debugger and ptrace

Byoungyoung Lee
Seoul National University
byoungyoung@snu.ac.kr
https://lifeasageek.github.io

Debugger

- A debugger is a computer program that is used to test and debugger other programs (the “target” program)

  - https://en.wikipedia.org/wiki/Debugger

GDB workflow

```
$ gdb /bin/ls
(gdb) run
(gdb) breakpoint *ADDRESS
(gdb) stepi
(gdb) continue
(gdb) info registers
(gdb) set $REG = VALUE
(gdb) x ADDRESS
(gdb) set *ADDRESS = VALUE
```

ptrace

- **ptrace**
  - A system call to perform: **p**rocess **trace**
  - One process (i.e., a tracer) traces other process (i.e., tracee)
    - Observe and control memory and registers of tracee
  - Used to implement breakpoint debugging and system call tracing
  - Used by GDB, strace, etc.

```
int main ( int argc, char * argv[] )
{
 int status;
 pid_t pid;
 struct user_regs_struct regs;
 int counter = 0;
 int in_call =0;

 switch(pid = fork()){
   case 0: /* in the child process */
     ptrace(PTRACE_TRACEME, 0, NULL, NULL);
     execvp(argv[1], argv+1);
   default: /* in the parent process */
     wait(&status);
     while(status == 1407){
       ptrace(PTRACE_GETREGS, pid, NULL, &regs);
       if(!in_call){
         printf(“SystemCall %ld called with %ld, %ld,     %ld\n”,regs.orig_rax, regs.rbx, regs.rcx,
regs.rdx);
         in_call=1;
         counter ++;
       }
       else in_call = 0;
       ptrace(PTRACE_SYSCALL, pid, NULL, NULL);
       wait(&status);
     } // end of while
   } // end of switch
   printf(“Total Number of System Calls=%d\n”, counter);
   return 0;
}
```

ptrace: request commands

```
#include <sys/ptrace.h>

long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

- **PTRACE_TRACE**
- **PTRACE_PEEKTEXT, PTRACE_PEEKDATA**
    - Read a word at addr (of tracee)
- **PTRACE_POKETEXT, PTRACE_POKEDATA**
    - Write a word at addr (of tracee)
- **PTRACE_GETREGS, PTRACE_SETREGS**
    - Copy/overwrite the tracee's registers
- **PTRACE_CONT**
    - Restart the stopped tracee process
- **PTRACE_SYSCALL, PTRACE_SINGLESTEP**
    - Restart the stopped tracee as for PTRACE_CONT
    - but arrange for the tracee to be stopped at the next entry to or exit from a system call, or after execution of a single instruction

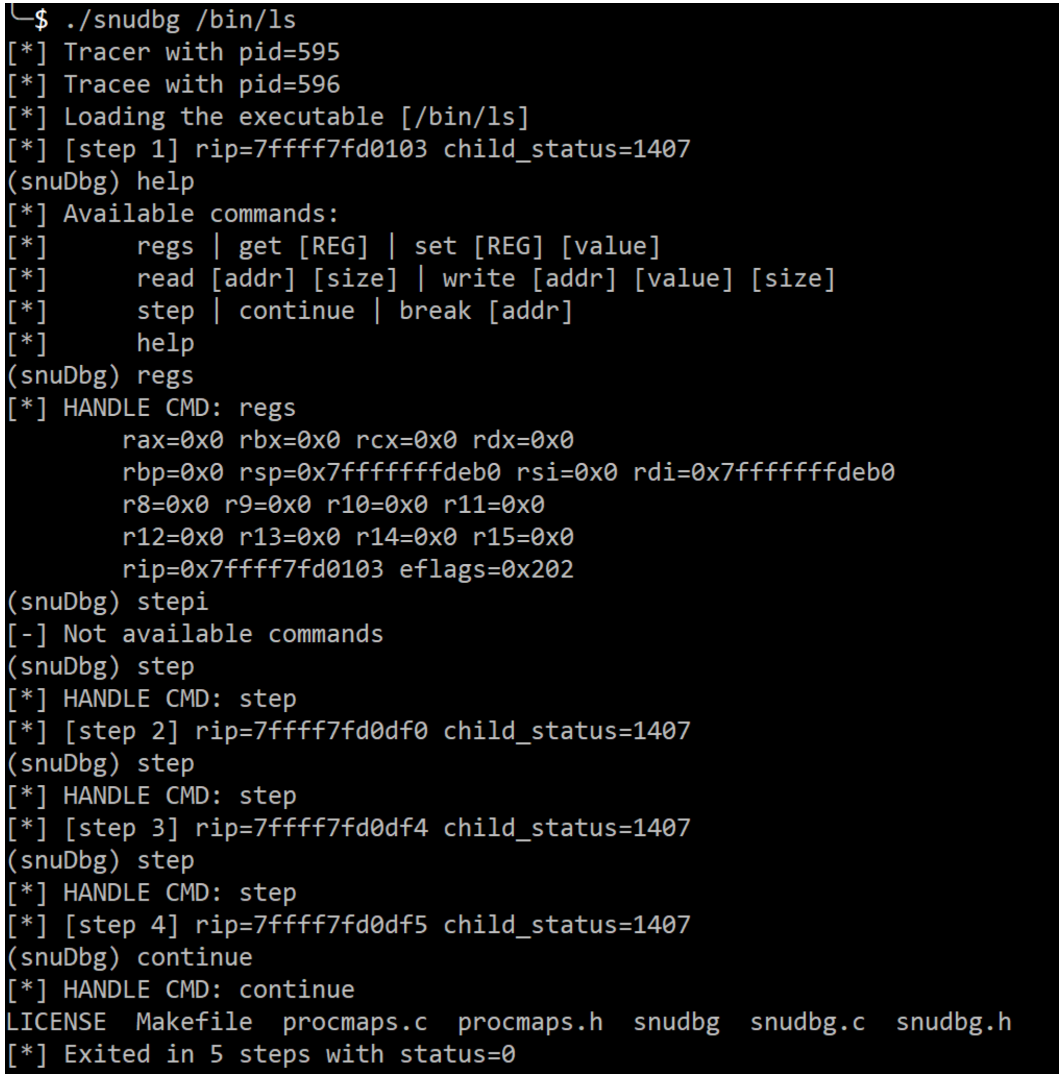
Breakpoints

- **Software breakpoints**
  - x86 instruction: int 3 (0xcc)
  - If CPU executes the instruction “int 3”, CPU raises an exception

- **Hardware breakpoints**
  - CPU reserves the debug register, from DR0-DR7
  - DR0 and DR7 hold the address to be used as a breakpoint

- **Memory breakpoints**
  - Typically implemented by changing the page permissions

Next Assignment: snuDBG



```
└$ ./snudbg /bin/ls
[*] Tracer with pid=595
[*] Tracee with pid=596
[*] Loading the executable [/bin/ls]
[*] [step 1] rip=7ffff7fd0103 child_status=1407
(snuDbg) help
[*] Available commands:
[*]      regs | get [REG] | set [REG] [value]
[*]      read [addr] [size] | write [addr] [value] [size]
[*]      step | continue | break [addr]
[*]      help
(snuDbg) regs
[*] HANDLE CMD: regs
        rax=0x0 rbx=0x0 rcx=0x0 rdx=0x0
        rbp=0x0 rsp=0x7fffffffdeb0 rsi=0x0 rdi=0x7fffffffdeb0
        r8=0x0 r9=0x0 r10=0x0 r11=0x0
        r12=0x0 r13=0x0 r14=0x0 r15=0x0
        rip=0x7ffff7fd0103 eflags=0x202
(snuDbg) stepi
[-] Not available commands
(snuDbg) step
[*] HANDLE CMD: step
[*] [step 2] rip=7ffff7fd0df0 child_status=1407
(snuDbg) step
[*] HANDLE CMD: step
[*] [step 3] rip=7ffff7fd0df4 child_status=1407
(snuDbg) step
[*] HANDLE CMD: step
[*] [step 4] rip=7ffff7fd0df5 child_status=1407
(snuDbg) continue
[*] HANDLE CMD: continue
LICENSE  Makefile  procmaps.c  procmaps.h  snudbg  snudbg.c  snudbg.h
[*] Exited in 5 steps with status=0
```

References

- ptrace(2) - Linux manual page: https://man7.org/linux/man-pages/man2/ptrace.2.html

- How do debuggers (really) work?: https://events.static.linuxfound.org/sites/events/files/slides/slides_16.pdf

- https://tldp.org/LDP/LG/issue81/sandeep.html

- GDB Internals Manual: https://sourceware.org/gdb/wiki/Internals