Assignment 4 A Dynamic Stack Dumping in Linux Kernel (100 points)

Exercise Objectives

1. To learn the basic programming technique in Linux kernel.
2. To practice the mechanisms and data structures of system call, and other Linux kernel objects

Project Assignment

Linux kernel dump_stack() function can give a call trace from process stack. To enable dump_stack() function, the kernel must be built with certain config options enabled. Then, dump_stack() function is inserted into source code and can get invoked at execution time.

In this assignment, you will implement two new syscalls to insert and remove dump_stack in the execution path of kernel programs. In other words, the insertion and deletion of dump_stack is done dynamically, without re-building the kernel. The basic technique is to use kprobe to invoke dump_stack() in a pre-handler. The two new syscalls should be defined as:

*SYSCALL_DEFINE2(insdump, const char __user *, symbolname, dumpmode_t, mode)*

SYSCALL_DEFINE1(rmdump, unsigned int, dumpid)

The syscall *insdump* inserts dump_stack operation at the symbol's address which enables back-tracing of the process stack when program execution hits the address. The process (normal or light-weight) which makes the syscall is regarded as the owner of the dump_stack operation and is responsible to make a call to remove the dump_stack operation (by calling the 2nd syscall *rmdump)*. The dump_stack operation is enabled for the owner process if dumpmode is 0, or for the processes that have the same parent process as the owner if dumpmode is 1. Otherwise, it is enabled for any processes.

The insertion of dump_stack operation can only be done at kernel symbols in the text section. On a successful insertion, the syscall returns a positive integer as dump_stack identifier (*dumpeid*) which should be used for *rmdump* syscall by the owner of the dump_stack operation. If the specified symbol cannot be found in kernel symbol table or is not in text section, *insdump* call should return EINVAL. Similarly, EINVAL should be returned by *rmdump* syscall if the remove operation fails. Note that a process may call *insdump* multiple times and multiple dump_stack operations can be inserted at a valid address. In addition, when a process terminated (exit or killed), all dump_stack operations inserted by the process should be removed.

The source code of your implementation for the syscalls should be named as "dynamic_dump_stack.c" (and "dynamic_dump_stack.h" if there is a header file). The source code must be saved in /lib directory. The insertion and deletion of dynamic dump stacks are enabled if configuration option *CONFIG_DYNAMIC_DUMP_STACK* is defined. Otherwise, the syscalls are no op's. Modifications in /lib/Makefile and /lib/Kconfig.debug may be needed to build the kernel correctly with the two new syscalls.

Since Linux system calls are statically defined and compiled into the kernel, you will need to rebuild the kernel with new system calls. You may consider the following suggested procedure to rebuild 3.19 Linux kernel for Galileo Gen2 board:

- Use the pre-built toolchain and the Linux kernel source linux3-19-r0.tar.bz2 from Dropbox/CSE438-530-Gen2_2017/Boot_image&SDK. Also, the ".config" file in the source code can be used for kernel building.
- Include the cross-compilation tools in your PATH:
  export PATH=path_to_sdk/sysroots/x86_64-pokysdk-linux/usr/bin/i586-poky-linux:$PATH

- Cross-compile the kernel
  ARCH=x86 LOCALVERSION= CROSS_COMPILE=i586-poky-linux- make -j4
- Build and extract the kernel modules from the build to a target directory (e.g ../galileo-install)
  ARCH=x86 LOCALVERSION= INSTALL_MOD_PATH=../galileo-install CROSS_COMPILE=i586-poky-linux- make modules_install
- Extract the kernel image (bzImage) from the build to a target directory (e.g ../galileo-install)
  cp arch/x86/boot/bzImage ../galileo-install/
- Install the new kernel and modules from the target directory (e.g ../galileo-install) to your micro SD card
  - Replace the bzImage found in the first partition (ESP) of your micro SD card with the one from your target directory (backup the bzImage on the micro SD card e.g. rename it to bzImage.old)
  - Copy the kernel modules from the target directory to the /lib/modules/ directory found in the second partition of your micro SD card.
- Reboot into your new kernel

Note that, after the first build, there is no need to rebuild/replace the kernel modules if you don't make any changes in the loadable modules. Also, to avoid building the kernel for each code revision, you may implement the functions in a loadable driver module and use device file operations to invoke them. Once the functions are fully developed, you can then add them as system calls and rebuild the kernel.

Due Date

The due date is 11:59pm, May 1.

What to Turn in for Grading

- The submission includes a **patch file** that consists of changes to the original kernel source code, a testing program, a Makefile (to make the testing program), and a README file. Compress the files into a zip archive named cse530-teamX-assgn04.zip. Note that any object code, original Linux kernel source code, or temporary build files should not be included in the submission. A submission that contains the original Linux kernel source code will not be graded and will receive a zero score. Please submit the zip archive to Blackboard by the due date and time.
- Please make sure that you comment the source files properly and the readme file includes a description about how to make and use your software. A sample result from your test run can be included in readme file. Don't forget to add your name and ASU id in the readme file.
- There will be 20 points penalty per day if the submission is late. Note that submissions are time stamped by Canvas. If you have multiple submissions, only the newest one will be graded. If needed, you can send an email to the instructor and TA to drop a submission.
- The assignment must be done individually. No collaboration is allowed, except the open discussion in the forum on Canvas. The instructor reserves the right to ask any student to explain the work and adjust the grade accordingly.
- Failure to follow these instructions may cause deduction of points.
- Here are few general rule for deductions:
  - No make file or compilation error -- 0 point for the part of the assignment.
  - Must have "–Wall" flag for compilation -- 5-point deduction for each warning.
  - 10-point deduction if no compilation or execution instruction in README file.
  - Source programs are not commented properly -- 10-20-point deduction.
- ASU Academic Integrity Policy (http://provost.asu.edu/academicintegrity), and FSE Honor Code (http://engineering.asu.edu/integrity) are strictly enforced and followed.