Kernel Module Project

An overview of some of the implementation details

Pranav Batra

IPC via /proc VFS files

- Read and write messages are marshalled into the appropriate structures. These structures allow for block-based interprocess communication between the userland program and the kernel module instead of a byte-oriented data stream.
- These same structures can be used with bidirectional multicast socket-based communication protocols (such as netlink or netlink-generic) instead of /proc files.
- Proc files are easier to work with

```
enum action {
    NOP, BLOCK, LOG, ENABLE_LOG_BLOCK, DISABLE_LOG_BLOCK, RESET_ALL
};

enum param {
    BLACK, WHITE, ADD_ADDR, REMOVE_ADDR, REMOVE_ALL, RESET
};

struct writemsg { //pod struct
    enum action action;
    enum param param;
    char addr[16];
};

struct readmsg { //pod struct
    char saddr[16];
    char daddr[16];
    int hook;
    bool blocked;
    struct timespec t;
};
```

Build tools

- Kbuild is the build system for used to compile the Linux kernel.
- Supports in-place builds however out-of-source builds are trickier
  - One solution involves using hardlinks
  - Clion only supports out-of-source compilation
- CMake is used to generate the makefile
- Objcopy converts the compiled module into an object file
  - Necessary to embed the module into the userland executable.

```
cmake_minimum_required(VERSION 2.6)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${CMAKE_SOURCE_DIR}/bin")
set(CMAKE_C_FLAGS "-Wall")
SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} --std=c++11")
add_executable(main main.cpp ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/netmod.ko.o)
add_custom_command(
        OUTPUT ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/netmod.ko.o
        COMMAND ./build_module.sh
        WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
        DEPENDS netmod.c header.h
)
```

```
#!/bin/sh
mkdir -p bin
ln -f netmod.c bin/netmod.c
cat > bin/Makefile << "EOF"
obj-m += netmod.o
ccflags-y := -I$(src)/../
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
EOF
cd bin && make 2>&1 | sed -r 's,^([^:]*/)bin/(../)?,\1,'
rm -f Makefile netmod.c
objcopy -I binary -O elf64-x86-64 -B i386 netmod.ko netmod.ko.o
```

Init_module, cleanup_module

- Called when the module is loaded and unloaded, respectively
- Register two netfilter hooks, NF_INET_LOCAL_IN for inbound packets and NF_INET_LOCAL_OUT for outbound ones.
- Set up the proc file with permissions 0666 – everyone is allowed to read from & write to the vfs file
- The name of the proc file is given in header.h: netmod_hello

```
int init_module(void) {
    in = template;
    in.hooknum = NF_INET_LOCAL_IN;
    in.priority = NF_IP_PRI_FIRST;
    nf_register_hook(&in);
    out = template;
    out.hooknum = NF_INET_LOCAL_OUT;
    out.priority = NF_IP_PRI_LAST;
    nf_register_hook(&out);
    proc_create(PROC_NAME, 0666, NULL, &proc_fops);
    printk("module loaded (%s)\n", path);
    return 0;
}

void cleanup_module(void) {
    nf_unregister_hook(&in);
    nf_unregister_hook(&out);
    remove_proc_entry(PROC_NAME, NULL);
    printk("module unloaded (%s)\n", path);
}
```

Ring buffers

- A ring buffer of readmsg's is used to store information about the logged packets
  - When the buffer is full, messages are overwritten in FIFO order – i.e., the oldest message are dropped first.
  - The index keep looping around, hence the name.
- Better to drop packets than to slow the system to a crawl!
  - This is what happens when you write to the log file directly from the kernel, my initial approach to logging.
- Bufsize is defined as 256 in header.h

```
if (log) {
    rbuf[ri].hook = state->hook;
    rbuf[ri].blocked = !allow;
    getnstimeofday(&rbuf[ri].t);
    snprintf(rbuf[ri].saddr, 16, "%pI4", &iph->saddr);
    snprintf(rbuf[ri].daddr, 16, "%pI4", &iph->daddr);
    ri++;
    ri = ri % bufsize;
    rl = min(rl + 1, bufsize);
}
```

```
rl--;
ri--;
if (ri < 0)
    ri += bufsize;
copy_to_user(buf, &rbuf[ri], count);
```

That’s all folks

- There is also a userland program that interfaces with the kernel module. It does some interesting stuff with regards to file descriptor manipulation and the use of fork/exec in order to overlay the kernel log file and the list of logged packets atop the CLI (command line interface).
- Take a look at the github source code for more information about the internals of both the userland program and the kernel module. Comments are included in the code!
- https://github.com/CS3281-2016/project