Linux LKM Firewall

v 0.95 (2/5/2010)

1 Overview

The learning objective of this project is for you to understand how firewalls work by designing and implementing a simple personal firewall for `Linux`. A personal firewall controls network traffic to and from a computer, permitting or denying communications based on the security policies set by an administrator.

Firewalls have several types; in this project, we focus on a very simple type, the *packet filter*. Packet filters act by inspecting packets; if a packet matches the packet filter's set of rules, the packet filter will either drop the packet or send an "error response" to its source. Packet filters are usually *stateless*; they filter each packet based only on the information contained in the current packet, without considering whether a packet may or may not be part of an existing stream of traffic. Packet filters often use a combination of source and destination address, port numbers, and protocol type to identify packets of interest.

2 Project Tasks

In this project, you will implement a packet filter for `Linux`. This firewall consists of two components: policy configuration and packet filtering.

2.1 Task 1: Firewall Policies

The policy configuration module is intended to allow system administrators to specify firewall policies. There are many types of policies that can be supported by personal firewalls, ranging from very simple to fairly complex. For your firewall, you must at least support the following policies, but you are encouraged (and will be rewarded) if your firewall can support more sophisticated policies. Your firewall should be able to block or unblock incoming and outgoing packets based on the following criteria:

1. *Protocol:* This specifies which protocol to target, e.g., TCP, UDP, or ICMP.
2. *Source and destination address:* Match packets that contain user-specified source and destination addresses. This includes address/netmask combinations, which are often used to block entire address ranges.
3. *Source and destination port number:* Match packets that contain user-specified source and destination port numbers.
4. *Action:* Specify an action to take when a packet matches a firewall rule. Common actions include
   - `BLOCK`: block packets.
   - `UNBLOCK`: used in conjunction with BLOCK to allow packets from just one address through while the entire network is blocked.

Configuration Tools. You need to implement a tool to allow the administrator to configure the firewall policies. Let us call this tool `fwadmin`. We give a few examples on how this tool can be used. However, feel free to change the syntax according to your own preference.

- `fwadmin --in --proto ALL --action BLOCK`
  Block all incoming packets.
- `fwadmin --in --proto TCP --action UNBLOCK`
  Allow only TCP incoming packets.
- `fwadmin --in --srcip 172.16.75.43 --proto ALL --action BLOCK`
  Block all the packets from the given IP address.
- `fwadmin --out --destip 172.20.33.22 --proto UDP --action UNBLOCK`
  Unblock the outgoing UDP packets if the destination is 172.20.33.22
- `fwadmin --in --srcip 172.16.0.0 --srcnetmask 255.255.0.0 --destport 80 --proto TCP --action BLOCK`
  Block all incoming TCP packets from the `172.16.0.0/16` network if the packets are directed towards port 80.
- `fwadmin --print`
  Print all rules.
- `fwadmin --delete 3`
  Delete the 3rd rule.
- `fwadmin --stats`
  Print statistics revealing the total number of packets blocked by your firewall per rule and in aggregate.

2.2 Task 2: Packet Filtering

The main part of firewall is the packet filter, which enforces the firewall policies set by an administrator. Since packet processing is done within the kernel, the filtering must also be done within the kernel. This requires us to modify the `Linux` kernel. In the past, this had to be done by modifying the kernel and rebuilding the entire kernel image. Modern `Linux` systems provides several mechanisms to facilitate packet manipulation that do not require the kernel image to be rebuilt. These two mechanisms are `Loadable Kernel Modules (LKMs)` and `netfilter`.

`LKMs` allow us to add a new module to the kernel at runtime. This new module enables us to extend the functionalities of the kernel without rebuilding the kernel or even rebooting the computer. You will implement the packet filtering portion of your firewall as an LKM. However, this is not enough. In order for your module to block incoming/outgoing packets, your module must be inserted into the packet processing pipeline. This can be accomplished using `netfilter`.

`netfilter` is designed to facilitate the manipulation of packets by authorized users. `netfilter` achieves this goal by implementing a number of *hooks* in the `Linux` kernel. These hooks are inserted into various places, including a packet's inbound and outbound paths. If we want to manipulate an incoming packet, we simply register a function in our module with a corresponding `netfilter` hook. Once an incoming packet arrives, the callback function we specified will be invoked. The function can then determine whether the packet should be blocked or not; moreover, it is possible to modify the packet in situ.

In this task, you will to use a `LKM` and `netfilter` to implement a packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. You should be able to support dynamic configuration, i.e., if an administrator alters the firewall policies while your module is installed, the module will immediately enforce the updated policies.

Policy Storage. Since your configuration tool runs in userspace, the tool has to send data to the kernel, where your packet filtering module resides. For performance reasons, the policies must be stored in kernel memory rather than a file.

3 Guidelines

3.1 Loadable Kernel Module

The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded and it prints out "Bye-bye World!" when the module is removed. The messages are not printed out on the screen; they are actually printed into the /var/log/kern.log file (among other locations).

```
#include <linux/module.h>
#include <linux/kernel.h>

/* we choose a GPL license to refrain from tainting the kernel */
MODULE_LICENSE("GPL");

/* other module information */
MODULE_DESCRIPTION("Hello-World Module");
MODULE_AUTHOR("Netsec Student");

int hello_init(void)
{
        printk(KERN_INFO "Hello World!\n");
        return 0;
}

void hello_exit(void)
{
        printk(KERN_INFO "Bye-bye World!.\n");
}

/* identify callbacks for module initialization and removal */
module_init(hello_init);
module_exit(hello_exit);
```

We now need to create Makefile to build the module. Place the following contents into your makefile (the program listed above is named hello.c). Then, type make and the above program will be compiled into a loadable kernel module.

```
obj-m += hello.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

```
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Once the module is built, you can use the following commands to load the module, list all installed modules, and remove the module:

```
 % sudo insmod mymod.ko        (inserting a module)
 % lsmod                       (list all modules)
 % sudo rmmod mymod.ko         (remove the module)
```

3.2 Interacting with your Loadable Kernel Module

For your firewall, the packet filtering part is implemented in the kernel, but the policy setting is done at the userspace. We need a mechanism to pass the policy information from a userspace program to the kernel module. There are several ways to do this; a standard approach is to use the proc filesystem, typically mounted at `/proc`. Please read the article from `http://www.ibm.com/developerworks/linux/library/l-proc.html` for detailed instructions. Once we set up a `/proc` file for our kernel module, we can use the standard `write()` and `read()` system calls to pass data to and from the kernel module.

3.3 A Simple Program that Uses Netfilter

Using `netfilter` is quite straightforward. All we need to do is to hook our functions (in the kernel module) to the corresponding `netfilter` hooks. Here we show a example that drops all IP packets:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>

/* we choose a GPL license to refrain from tainting the kernel */
MODULE_LICENSE("GPL");

/* other module information */
MODULE_DESCRIPTION("Linux LKM Firewall");
MODULE_AUTHOR("Netsec Student");

/* netfilter struct used for registering a hook */
struct nf_hook_ops hook_options;

/* our callback function for the netfilter hook */
unsigned int process_packet(unsigned int hooknum,
                            struct sk_buff *skb,
                            const struct net_device *in,
                            const struct net_device *out,
                            int (*okfn)(struct sk_buff *))
{
```

```
        /* drop packet */
        return NF_DROP;

}

/* module initialization function */
int firewall_init(void)
{

        /* log message -- check for messages in /var/log/kern.log */
        printk(KERN_INFO "initializing firewall module!\n");

        /* identify callback function for hook */
        hook_options.hook = process_packet;

        /* indicate where to hook a packet in processing pipeline */
        hook_options.hooknum = NF_INET_PRE_ROUTING;

        /* specify the protocol we want to hook */
        hook_options.pf = PF_INET;

        /* set the priority of our hook */
        hook_options.priority = NF_IP_PRI_FIRST;

        /* register hook with netfilter */
        nf_register_hook(&hook_options);

        return(0);

}

void firewall_exit(void)
{

        /* unregister hook with netfilter when removing module */
        nf_unregister_hook(&hook_options);

        printk(KERN_INFO "removing firewall module!\n");

}

/* identify callbacks for module initialization and removal */
module_init(firewall_init);
module_exit(firewall_exit);
```

3.4 The `iptables` program

`Linux` has a tool called `iptables`, which is essentially a firewall built upon the `netfilter` mechanism. You can consider your firewall as a mini-version of `iptables`. You are encouraged to play with `iptables` to gain some inspiration for your own design. However, copying the code from `iptables` is strictly forbidden. Moreover, you may find some tutorials that provide sample code for simple firewalls. You can learn from those tutorials and play with the sample code, but you have to write your own code. In your project report, you must submit your code and explain its key parts.

3.5 Parsing Command Line Arguments

We strongly recommend that you use the `getopt` library to parse command line arguments in your userspace programs. Please read the tutorial located at the following URL:

`http://www.gnu.org/s/libc/manual/html_node/Getopt.html`

3.6 Environment

We recommend using Ubuntu 9.10 Desktop and VirtualBox while developing this project. The Ubuntu installation has all of the tools you'll need to begin installed by default. Ubuntu 9.10 can be found at `http://www.ubuntu.com/` and VirtualBox can be downloaded from `http://www.virtualbox.org/`.

3.7 Additional resources

In addition to the URLs provided previously in this document, you may wish to reference the *Linux Device Drivers* book located online: `http://www.makelinux.net/ldd3/`. The `netfilter` website also has a lot of great information: `http://www.netfiler.org/`. You may find it helpful to search through code in the `Linux` kernel, e.g., for structure definitions, as well. One online resource for doing so is located here: `http://lxr.linux.no/`

4 Deliverables, Deadline, and Other Information

This project is meant to be completed by groups of 2 or 3 students. Under some circumstances I may consider a group of 1 or 4; however, you must receive my explicit permission to do so. If you choose to do the project by yourself, you will be evaluated no differently than a group of 2 or 3. If you choose to do the project in a group of 4, I expect your project to be even more impressive than that of a normal-sized group!

When you have completed the assignment, create an archive of your source code, makefile, and design document as well any additional instructions that may be useful in evaluating your firewall. If your code does not compile, we will not evaluate it. This project is due no later than 11:59PM on Friday, Februrary 19th. Submit your project as an attachment to Nitesh via email (*jhunetsec.ta@gmail.com*).

5 Version History

v 0.95 (2/5/2010) – The current version of this document. Added due date and deliverable information.
v 0.9 (2/4/2010) – The original version of this document.

6 Attribution

These project specifications are based off of a document written by Wenliang Du. What follows is his original copyright notice: