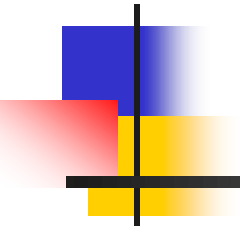


Firewall

Lecture 11

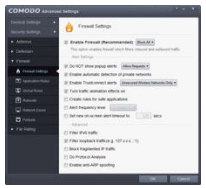
Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.

Introduction

- ***Firewall***: stop *unauthorized traffic* flowing from one side to another side (e.g., network)
  - can be deployed "*anywhere*"
  - *separating* ***trusted*** and ***untrusted*** components of networks
  - *differentiating* sub-networks ***within*** a trusted network
    - create *distinction* between various divisions within organization
- Firewall implementation: *hardware*, *software*, or *combination*
- ***Firewall's main functionalities***:
  - *filtering data*
  - *redirecting traffic*
  - *protecting against network attacks*



Firewall Requirements

- Well-designed firewalls meet following ***requirements***
  1. all traffic between two *trust zones* should pass through
     - very tough challenge within large network
  2. only authorized traffic (defined by *security policy*) should be allowed to pass through
  3. immune to penetration which implies using a hardened system with secured OS
     - firewall itself must be *robust* and *comprehensive*
       - include IDS and IPS
       - regular update
       - access control
         - who can access to the setting of firewall
       - etc.

Firewall Policy

- ***Firewall policy***: *rules* that should be *enforced*
  - ***rule***: provide controls for traffic on network
  1. *user control*: controls access to the data based on ***the role of the user*** who is attempting to access it
     - applied to user inside firewall perimeter
  2. *service control*: access is controlled ***by the type of service*** offered by the host that is being protected by firewall
     - enforced on network address, port number, protocol
  3. *direction control*: determines the ***direction*** in which requests may be initiated and are allowed to flow through the firewall
     - inbound & outbound traffic

Firewall Actions

- ***Firewall actions***:
  - *accepted*: allowed to enter through firewall
  - *denied*: not permitted to enter through firewall
  - *rejected*: similar to *denied*, but notifying the source of packet about decision via ICMP packet
- firewall inspects traffic from *both directions*

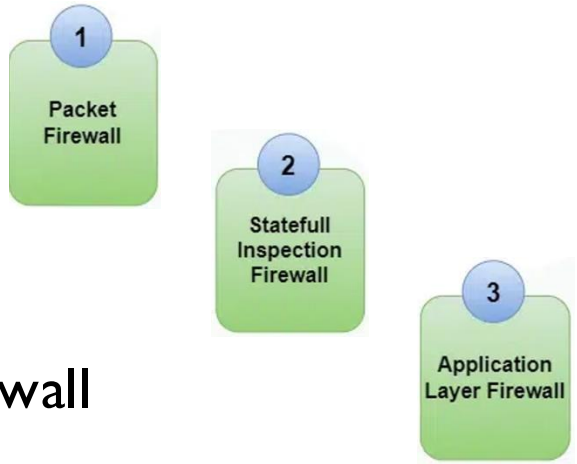
ingress filtering: inspects the incoming traffic to safeguard an internal network and prevent attacks from outside.

egress filtering: inspects the outgoing network traffic and prevent the users in the internal network to reach out to the outside network.

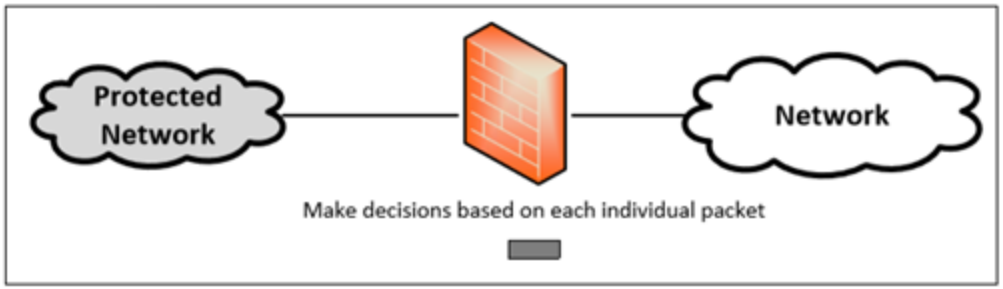
- for example:
  - blocking social networking sites in school

Types of Firewalls

- Depending on the *mode* of operation, there are three types of firewalls
  1. packet filter firewall
  2. stateful firewall
  3. application/proxy firewall



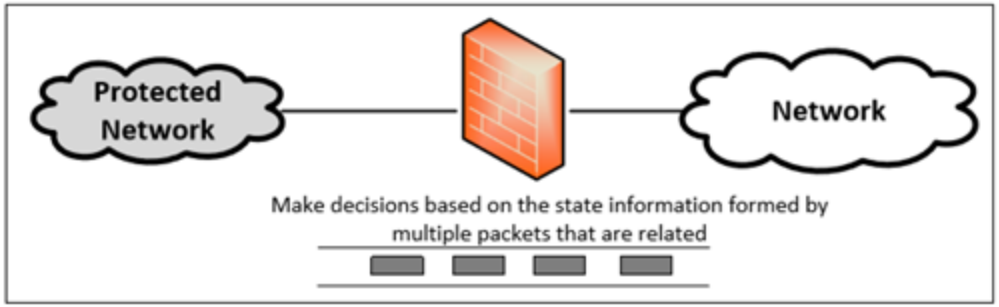
Packet Filter Firewall



How it works: controls traffic based on the information in *packet headers*, without looking into the payload that contains application data

- Inspects each packet and make decision based on information in the packet header
- Doesn't pay attention to if the packet is a part of existing stream or traffic
- Advantages:
  - ***speed***; doesn't maintain the states about packets
    - also called *stateless firewall*

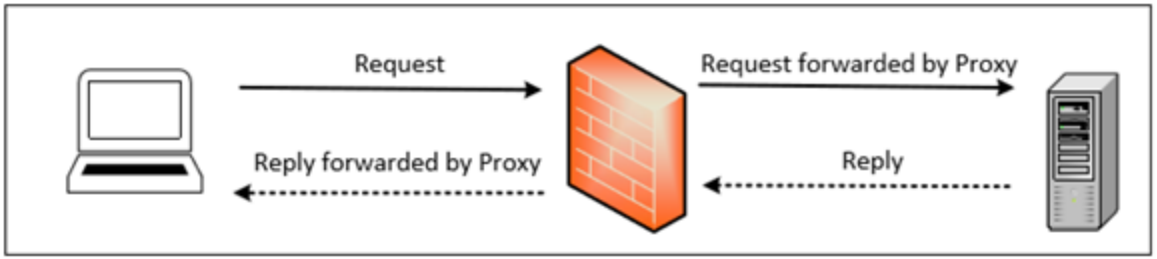
Stateful Firewall



How it works: tracks the state of traffic by monitoring all the connection interactions until is closed

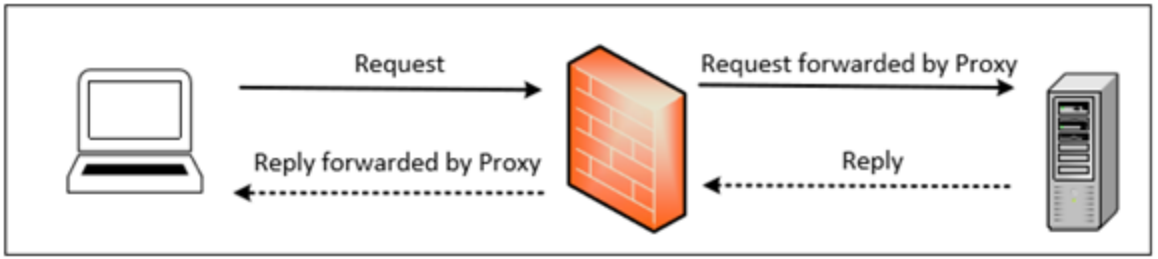
- Retrains packets until decision can be made
  - connection state table is maintained to understand the context of packets
- Some of them also inspect app. data for well-known protocols
  - identify and track related connections among all interactions
- Advantages:
  - allowing through traffic that belong to existing connection

Application/Proxy Firewall



- Controls input, output, and access from/to application or service
- How it works: unlike packet/stateful firewalls, inspects network traffic up to application layer (payload)
- Typical implementation: proxy (application proxy firewall)
  - intermediary: "*impersonating*" the intended recipient
    - client's connection terminates at proxy
      - a new connection initiated from proxy to destination
    - data is analyzed up to application layer to determine if the packet should be allowed or rejected
      - protecting internal from risk of direct interaction
      - protecting sensitive information being leaked

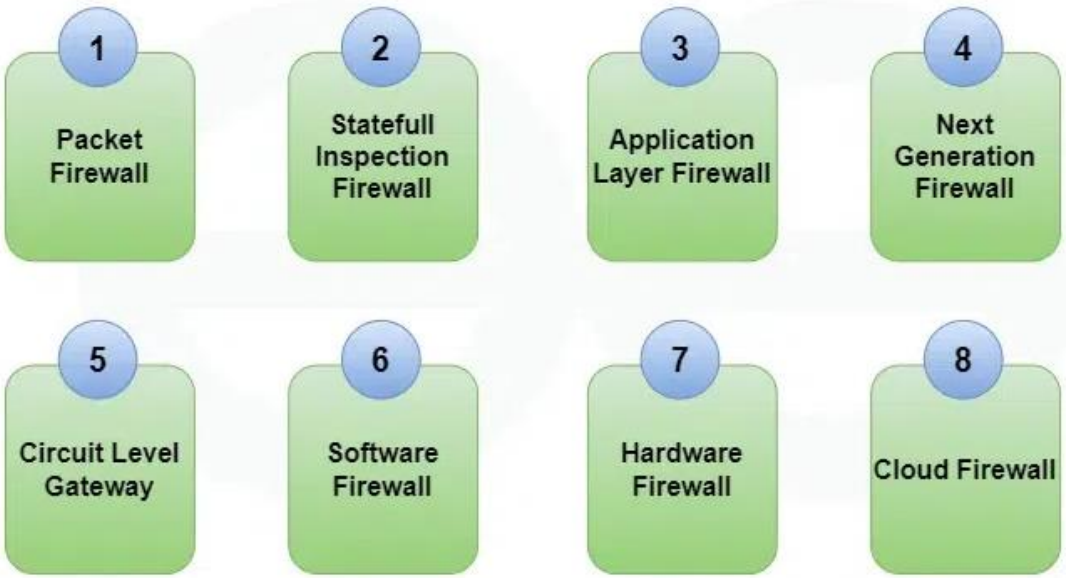
Application/Proxy Firewall



- Limitation:
  - implementing new proxies for new protocols
  - slower (reading the entire packet)
- Advantages:
  - authenticate user directly rather than depending on network address of system

Types of Firewalls (cont.)

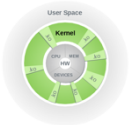
- Depending on the *mode* of operation, there are three types of firewalls
  1. packet filter firewall
  2. stateful firewall
  3. application/proxy firewall



4. Next-generation Firewalls
   - includes additional features like app. awareness and control, integrated intrusion prevention
5. Circuit-level Gateways
   - provides UDP and TCP connection security and works between an OSI network model's transport and application layers
6. Software Firewall
   - protects computers from any external attacks such as unauthorized access, malicious attacks, etc.
7. Hardware Firewall
8. Cloud Firewall

Building Firewall using Netfilter

- Packet filter firewall implementation in Linux
  - packet filtering can be done inside the kernel
    - need to modify the kernel
- Linux provides two mechanisms (no need to recompile kernel)



Loadable Kernel Modules: allow privileged users to dynamically add/remove modules to the kernel, so there is no need to recompile the entire kernel

Netfilter: provides hooks at critical points on the packet traversal path inside Linux kernel

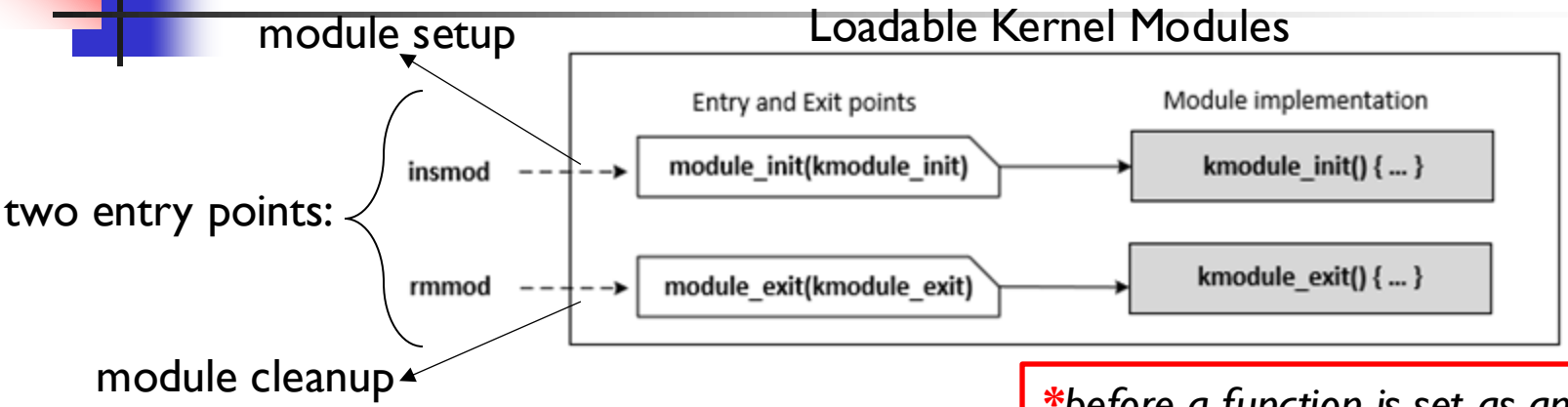
- allow packets to go through additional program logics (e.g., packet filtering program)

Writing Loadable Kernel Modules

- Modular Linux kernel
  - a minimal part of kernel is loaded into memory
- Additional features can be implemented as kernel modules, and be loaded into kernel dynamically
  - e.g., a new kernel module supporting a new hardware
- ***Kernel module***: pieces of code that can be loaded and unloaded on-demand at runtime
  - they don't run as specific processes but are executed in kernel on behalf of current process
  - a process needs ***root privilege*** or ***CAP_SYS_MODULE*** capability to be able to insert or remove kernel modules

    ref.: https://man7.org/linux/man-pages/man7/capabilities.7.html

Loadable Kernel Modules (cont.)



before a function is set as an entry point, it must first be defined in the program

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

static int kmodule_init(void) {
   printk(KERN_INFO "Initializing this module\n");
   return 0;
}

static void kmodule_exit(void) {
   printk(KERN_INFO "Module cleanup\n");
}

module_init(kmodule_init);          ①
module_exit(kmodule_exit);          ②

MODULE_LICENSE("GPL");
```

defining function

printk(): print to kernel log buffer

setup entry point

cleanup entry point

specify an initialization function that will be invoked when the kernel module is inserted.

specify a cleanup function that will be invoked when the kernel module is removed.

Compiling Kernel Modules

The easiest way to build a kernel module is to use ***makefile***

```
obj-m += kMod.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
```

object file to be built

specifies object files which are built as loadable kernel modules

Makefile

-M: signifies that an external module is being built and tells the build environment where to place the built module file once it is built

-C: specify the directory of the library files for the kernel source

```
$ make
make -C /lib/modules/3.5.0-37-generic/build
    M=/home/seed/labs/firewall/lkm modules
make[1]: Entering directory '/usr/src/linux-headers-3.5.0-37-generic'
  CC [M]  /home/seed/labs/firewall/lkm/kMod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/labs/firewall/lkm/kMod.mod.o
  LD [M]  /home/seed/labs/firewall/lkm/kMod.ko
make[1]: Leaving directory '/usr/src/linux-headers-3.5.0-37-generic'
```

compiled as points to be loaded into the kernel at runtime

compile C file into object file

linking compiled object files together

Installing Kernel Modules

insert modules into the kernel

```
// Insert the kernel module into the running kernel.
$ sudo insmod kMod.ko
```

display the status of modules in the Linux kernel

```
// List kernel modules
$ lsmod | grep kMod
kMod                   12453  0

// Remove the specified module from the kernel.
$ sudo rmmod kMod
```

filter the output with grep

remove a module from the kernel

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

static int kmodule_init(void) {
   printk(KERN_INFO "Initializing this module\n");
   return 0;
}
static void kmodule_exit(void) {
   printk(KERN_INFO "Module cleanup\n");
}

module_init(kmodule_init);        ①
module_exit(kmodule_exit);        ②

MODULE_LICENSE("GPL");
```

examine the kernel log buffer and print the message buffer of kernel

```
$ dmesg
......
[65368.235725] Initializing this module
[65499.594389] Module cleanup
```

- in the sample code, we use printk() to print out messages to the kernel buffer
- we can view the buffer using dmesg